1 Einleitung

In diesem Handbuch sind alle Assemblerbefehle aus dem Befehlsvorrat der vom Betriebssystem BS2000 unterstützten Zentraleinheiten einzeln und ausführlich beschrieben.

Die Assemblerbefehle sind diejenigen Befehle des Befehlsvorrats, die von Anwendern ohne Einschränkungen zur Formulierung von Assemblerprogrammen benutzt werden können. Die Befehle sind außerdem "sicher" in dem Sinne, daß bei ihrer Ausführung der volle Schutz der Hardware und des Betriebssystems BS2000 gewahrt ist.

1.1 Zielgruppe

Das Handbuch richtet sich an Anwender, die im BS2000 Programme in der Assembler- oder Makrosprache erstellen, benutzen oder warten. Assembler- und Betriebssystem-Grundkenntnisse werden vorausgesetzt.

1.2 Konzept des Handbuchs

Die Beschreibung der Assemblerbefehle folgt einem einheitlichen Schema. Bei jedem Befehl sind explizit dargestellt:

- seine Funktion
- sein Assemblerformat, d.h. seine Schreibweise in Assemblersprache
- sein Maschinenformat, d.h. seine Darstellung in der Zentraleinheit
- sein Ablauf im Detail
- etwaige von ihm gesetzte Werte der Anzeige
- sowie die bei seinem Ablauf möglichen Programmunterbrechungen

Außerdem haben wir die meisten Befehle ergänzt um

- Programmierhinweise sowie
- ein oder mehrere Beispiele.

Die Befehle selbst sind in 4 Gruppen gegliedert:

- Allgemeine Befehle (Kapitel 3)
- Dezimalbefehle (Kapitel 4)
- Gleitpunktbefehle (Kapitel 5)
- ESA-Befehle (Kapitel 6)

Innerhalb dieser Gruppen sind die Befehle alphabetisch nach ihrer mnemotechnischen Bezeichnung geordnet.

Das Kapitel 2 enthält grundlegende Erläuterungen.

1.3 Änderungen gegenüber dem Vorgänger-Handbuch

Beschreibung der ESA-Unterstützung im Kapitel 2 (2.1.3, 2.1.4, 2.2.2) und im neuen Kapitel 6 (ESA-Befehle).
Befehlslisten im Anhang 7.2 und 7.3 mit den ESA-Befehlen ergänzt.

Der Zugriff auf gemeinsam benutzte Daten in Multiprozessor-Anlagen ist im Anhang 7.6 beschrieben.

2 Grundlagen

2.1 Hauptspeicher-Adressierung

Der Hauptspeicher kann als Folge einzelner Bit betrachtet werden. Diese Folge ist in Einheiten von jeweils 8 Bit unterteilt, die Byte genannt werden. Jedem Byte ist eine eindeutige, ganze Zahl zugeordnet, genannt Adresse, die es identifiziert. Aufeinanderfolgende Byte haben aufeinanderfolgende Adressen. Der Wertebereich der Adressen, der Adreßraum, beginnt bei 0 und endet bei einer systemspezifischen Obergrenze.

2.1.1 Virtuelle Adressen

Alle in diesem Handbuch beschriebenen Assemblerbefehle benutzen ausschließlich sog. **virtuelle** Adressen und verarbeiten nur Operanden, deren Adressen virtuell sind. Auch die Befehle selbst sind virtuell adressiert. In den Zentraleinheiten selbst werden jedoch noch zwei weitere Arten von Adressen unterschieden, nämlich absolute und reale Adressen. Sie werden aber nur unterhalb der Oberfläche der Assemblerbefehle verwendet und sind in einem Anwenderprogramm weder sichtbar noch zu beeinflussen.

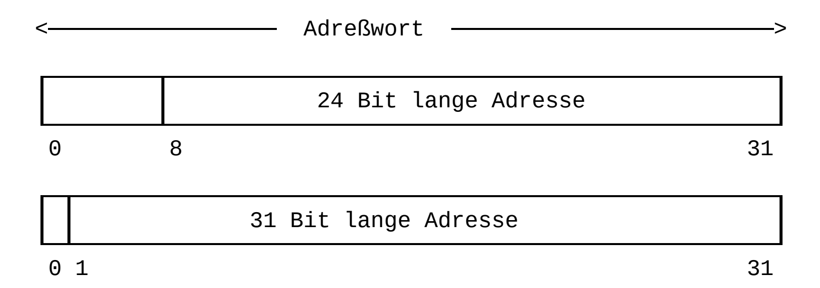
Virtuelle Adressen werden vom Betriebssystem beim Laden eines Programms oder auf dessen explizite Anforderung für ein Programm exklusiv **bereitgestellt** (engl. allocated). Die Bereitstellung erfolgt in Vielfachen von **Seiten**; Seiten sind Adreßbereiche von 4096 (2^{12}) Byte Länge, deren Anfangsadresse durch 4096 ohne Rest teilbar ist. Ein Programm darf nur auf (für es) bereitgestellte virtuelle Adressen (lesend oder schreibend) zugreifen. Wenn es auf eine nicht bereitgestellte Adresse zuzugreifen versucht, etwa aufgrund eines Programmierfehlers, dann wird dieser Zugriff nicht ausgeführt und es erfolgt eine Programmunterbrechung wegen Adreßumsetzungsfehlers (siehe 2.4).

2.1.2 24 Bit lange und 31 Bit lange Adressen

Adressen treten in zwei Längen auf, als 24 Bit lange Adressen und als 31 Bit lange Adressen. Eine 24 Bit lange Adresse kann 16 777 216 (16 Mega-) Byte des (virtuellen) Adreßraums identifizieren, eine 31 Bit lange Adresse 2 147 483 648 (2 Giga-) Byte.

Sowohl 24 Bit lange wie auch 31 Bit lange Adressen werden als sog. **Adreßwort** rechtsbündig in 4 Byte langen Hauptspeicherbereichen oder Mehrzweckregistern bereitgestellt. Wenn es nicht anders beschrieben ist, werden dabei die 8 Bit links von einer 24 Bit langen Adresse und das eine Bit links von einer 31 Bit langen Adresse mit 0 besetzt.

Die Bitstellen einer 24 Bit langen Adresse werden von 8 bis 31, die einer 31 Bit langen Adresse von 1 bis 31 numeriert:



2.1.3 Adressierungsmodi

Die Länge von Adressen und damit die Größe des Adreßraums, den ein Anwenderprogramm benützen kann, wird durch den **Adressierungsmodus** festgelegt. Es gibt zwei Adressierungsmodi, den 24-Bit-Adressierungsmodus und den 31-Bit-Adressierungsmodus. Zu jedem Zeitpunkt befindet sich eine Zentraleinheit in einem dieser Modi und verwendet oder erzeugt dementsprechend entweder 24 Bit lange oder 31 Bit lange Adressen. Als Folge davon verhalten sich diejenigen Assemblerbefehle, die Adressen explizit bereit stellen (wie z.B. der Befehl LA), unterschiedlich je nach dem Adressierungsmodus, in dem sie ausgeführt werden. Bei solchen Befehlen sind in diesem Handbuch diese Unterschiede im Einzelnen dargestellt.

Ab BS2000 V11.0 wird ein **neuer Adressierungsmodus** zur Erweiterung des virtuellen Adreßraums unterstützt, der **AR-Modus** (access register mode); siehe Kapitel 6, AR-Modus. Durch eine neue Hardware (ESA-Anlagen) wird die Möglichkeit geschaffen, mehr virtuelle Adreßräume für Daten zu nutzen. Auf diesen sogenannten **ESA-Anlagen** (Enterprise System Architecture) werden neben dem bisherigen Adreßraum, der jetzt **Programmraum** (program space) heißt, weitere Adreßräume für Daten, sogenannte **Datenräume** (data spaces) zur Verfügung gestellt.
Um die Möglichkeit der Adreßraumerweiterung zu nutzen, muß im Programm (dem BS2000) mitgeteilt werden (siehe Befehl SAC, Kapitel 6), daß es mit dem zusätzlichen Registersatz (den Zugriffsregistern, siehe 2.2.2 und Kapitel 6) arbeiten, d.h. im AR-Modus laufen soll.

2.1.4 Befehlsadressen, Befehlsfolgeadressen

Ein Anwenderprogramm besteht gewöhnlich aus Befehlen und Daten. Beide sind im Hauptspeicher gespeichert und beide haben (virtuelle) Adressen. Die Adresse eines Befehls, die **Befehlsadresse**, ist die Adresse seines ersten Byte; dieses Byte enthält bei allen Befehlen den sog. Operationscode. Bei jeder Ausführung eines Befehls wird von der Zentraleinheit die **Befehlsfolgeadresse** ermittelt; das ist die um die Länge (in Byte) des momentanen Befehls erhöhte Befehlsadresse. Wenn der momentane Befehl kein Sprungbefehl ist, dessen Sprungbedingung erfüllt ist, dann wird nach der Befehlsausführung des momentanen Befehls die Befehlsfolgeadresse zur Befehlsadresse gemacht und demzufolge bei dem folgenden Befehl fortgefahren, andernfalls bei der Adresse, die im Sprungbefehl als Sprungadresse bestimmt ist. Auch wenn der AR-Modus eingeschaltet ist, liegt die Befehlsadresse immer im Programmraum. Bei einem Sprungbefehl wird das entsprechende Zugriffsregister nicht ausgewertet, es ist also nicht möglich in einen Datenraum zu springen.

2.1.5 Operandenadressen, Adreßberechnung

Alle Befehle entnehmen ihre Operanden entweder Registern oder dem Hauptspeicher bzw. schreiben sie dorthin. Im Falle von Register-Operanden wird die entsprechende Registernummer in einem R-Feld der Operandenadresse bestimmt. Im Falle von Hauptspeicher-Operanden wird die Hauptspeicheradresse aus zwei, bei den RX-Befehlen aus drei Komponenten berechnet: aus der **Basisadresse**, der **Distanzadresse** und -ggf.- der **Indexadresse**.

Die Basis- und Indexadressierung ermöglicht indirekten ("pointer"-) Zugriff auf Operanden, die Distanzadressierung ermöglicht die Adressierung relativ zu einer Basis- oder Indexadresse. Die Basisadressierung dient insbesondere der Verschiebbarkeit eines Programmteils im Adreßraum eines Anwenderprogramms. Als Basisadresse verwendet man meist die Anfangsadresse eines größeren Bereichs aus (logisch zusammengehörigen) Daten oder Befehlen, als Distanzadresse dient dann der Abstand eines einzelnen Elements vom Anfang dieses Bereichs (bis zu 4095 Byte). Die bei den RX-Befehlen zusätzlich mögliche Indexadressierung erlaubt doppelt indirekten Zugriff auf Operanden, etwa auf die Elemente einer Tabelle in einer Tabelle.

Die effektive Adresse eines Hauptspeicher-Operanden wird berechnet als Summe

- aus der Basisadresse, d.h. aus der 32 Bit langen Binärzahl in dem Mehrzweckregister, das durch das B-Feld einer Operandenadresse bestimmt ist (Basisregister),
- aus der Distanzadresse, d.h. aus der 12 Bit langen Binärzahl, die direkt im D-Feld einer Operandenadresse angegeben ist und

- aus der Indexadresse, d.h. aus der 32 Bit langen Binärzahl in dem Mehrzweckregister, das durch das X-Feld einer Operandenadresse bestimmt ist (Indexregister). Eine solche Indexadresse ist allerdings nur bei den sog. RX-Befehlen vorhanden.

Die Summanden werden als Binärzahlen ohne Vorzeichen behandelt, ein etwaiger Übertrag über die höchstwertige Binärstelle wird ignoriert. Die Summe wird je nach Adressierungsmodus auf die niedrigstwertigen 24 Bit oder 31 Bit gekürzt und die oberen acht Bit bzw. das obere eine Bit werden auf 0 gesetzt. Das Resultat ist dann die (virtuelle) Adresse des Operanden und zwar in den meisten Fällen die Adresse seines ersten (höchstwertigen) Byte.

Wenn das B-Feld oder das X-Feld einer Operandenadresse (oder beide) gleich Null sind, wird die entsprechende Komponente nicht in die Summierung einbezogen. Es ist demzufolge nicht möglich, das Mehrzweckregister 0 zur Basis- oder Indexadressierung zu benutzen.

Wenn der **AR-Modus** (siehe 2.1.3 und Kapitel 6) eingeschaltet ist, berechnet sich eine effektive Adresse wie bisher (Basisadresse + Distanzadresse + Indexadresse), jedoch unter Berücksichtigung des Zugriffsregisters (siehe 2.2.2 und Kapitel 6).

2.1.6 Ausrichtung auf Halbwort-, Wort- und Doppelwortgrenzen

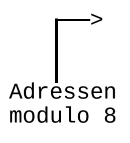
Neben den Hauptspeicher-Elementen Bit und Byte werden noch die Hauptspeicher-Elemente **Wort**, **Halbwort** und **Doppelwort** benutzt.

Ein Halbwort ist ein 2 Byte langer Hauptspeicherbereich mit einer durch 2 ohne Rest teilbaren, also geradzahligen Anfangsadresse. Dementsprechend ist ein Wort ein 4 Byte langer Hauptspeicherbereich, dessen Anfangsadresse durch 4 ohne Rest teilbar ist und ein Doppelwort ist ein 8 Byte langer Hauptspeicherbereich mit einer durch 8 ohne Rest teilbaren Anfangsadresse. Umgekehrt wird eine durch 2 bzw. 4 bzw. 8 ohne Rest teilbare Adresse gewöhnlich "Halbwortgrenze" bzw. "Wortgrenze" bzw. "Doppelwortgrenze" genannt.

Alle Befehle müssen in allen Zentraleinheiten an einer Halbwortgrenze ausgerichtet sein, diese Ausrichtung wird automatisch vom Assembler [1] ausgeführt. Darüberhinaus müssen bei vielen Befehlen die Operanden einer Ausrichtungsbedingung genügen. Zwar ist diese zweite Forderung abhängig von der Zentraleinheit (bei "älteren" Zentraleinheiten führt eine fehlende Operandenausrichtung z.B. bei dem Befehl L zu einer Programmunterbrechung mit dem Gewicht ’5C’, während die "neueren" Zentraleinheiten diese Operanden verarbeiten), aber sie ist in diesem Handbuch so dargestellt, wie sie in der restriktivsten Zentraleinheit gilt. Aus Performancegründen wird empfohlen, auch in Programmen, die auf "neueren" Anlagen laufen, die Operanden auf die entsprechenden Grenzen auszurichten (siehe Handbuch "ASSEMBH, Beschreibung"). Dadurch ist ein Assemblerprogramm, das die hier beschriebenen Ausrichtungsbedingungen der Operanden erfüllt, maximal portabel.

Wenn ein Befehl voraussetzt, daß einer seiner Operanden in einem Halbwort oder Wort oder Doppelwort enthalten ist, dann sagt man, daß dieser Operand "ausgerichtet" sein müsse und zwar je nach Fall "an" einer Halbwortgrenze bzw. Wortgrenze bzw. Doppelwortgrenze. Zum Beispiel verlangen alle Befehle für Binärzahlen, daß die Hauptspeicherbereiche für diese Binärzahlen ausgerichtet sind. Die Ausrichtungsbedingung gilt auch für alle Gleitpunktzahlen im Hauptspeicher, aber nicht für Dezimalzahlen und Zeichenfelder.

Es ergibt sich folgende Hauptspeicherstruktur:



0	1	2	3	4	5	6	7
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
Halbwort		Halbwort		Halbwort		Halbwort	
Wort				Wort			
Doppelwort							

2.2 Register

Fast alle Befehle verlangen, daß ein oder mehrere ihrer Operanden in einem "Register" enthalten sind. Register sind vom Hauptspeicher unabhängige Speicherbereiche mit sehr schnellem Zugriff. Es gibt drei Arten von Registern, nämlich Mehrzweckregister, Zugriffsregister und Gleitpunktregister.

2.2.1 Mehrzweckregister

Die Zentraleinheiten verfügen über 16 Mehrzweckregister, numeriert von 0 bis 15. Jedes Register ist 32 Bit lang, kann also 4 Byte oder ein Wort aufnehmen. Die Mehrzweckregister dienen zur Basis- und Index-Adressierung oder als Akkumulatoren in arithmetischen Operationen. Wenn sie als Akkumulatoren verwendet werden, werden sie in Befehlen durch die explizite Angabe ihrer Registernummer in einem R-Feld bestimmt, allerdings bestimmen einige Befehle, z.B. der Befehl TRT, die von ihnen verwendeten Mehrzweckregister implizit.

Manche Befehle verwenden als Operanden die Inhalte zweier aufeinanderfolgender Mehrzweckregister. (Diese werden dann ein **Mehrzweckregister-Paar** genannt). Das erste der beiden Register (mit dem höherwertigen Teil des Operanden) muß dann stets ein geradzahliges Mehrzweckregister sein und das zweite Register (mit dem niederwertigen Teil des Operanden) ist dann das nachfolgende ungeradzahlige Mehrzweckregister. Im R-Feld für solche Operanden wird das geradzahlige Register angegeben.

Die Mehrzweckregister können auch zur Basis- oder Index-Adressierung von Operanden verwendet werden. In diesen Fällen bestimmt das B- bzw. das X-Feld einer Operandenadresse das verwendete Register. Ein Wert 0 in einem B- oder X-Feld bestimmt allerdings nicht das Mehrzweckregister 0 als Basis- oder Indexregister, sondern legt fest, daß *keine* Basis- bzw. Index-Adressierung bei der Adreßberechnung der effektiven Operandenadresse erfolgen soll. Deshalb kann das Mehrzweckregister 0 nicht als Basis- oder Indexregister verwendet werden.

2.2.2 Zugriffsregister

Die ESA-Anlagen verfügen über 16 Zugriffsregister (AR, access register), numeriert von 0 bis 15. Jedes Register ist 32 Bit lang, kann also 4 Byte oder ein Wort aufnehmen. Die Zugriffsregister dienen dem Zugriff auf die Datenräume (data spaces).
Die 16 Zugriffsregister (AR) sind den 16 Mehrzweckregistern (MZR) eindeutig zugeordnet.
Enthält das B-Feld (Basisregister) einer Operandenadresse den Wert 0, so wird weder das Mehrzweckregister 0 zur Adressberechnung der effektiven Operandenadresse herangezogen noch das Zugriffsregister 0 zur Adressierung eines Datenraums.

2.2.3 Gleitpunktregister

Die Zentraleinheiten verfügen über 4 Gleitpunktregister, die ausschließlich durch Gleitpunktbefehle angesprochen und auch nur für Gleitpunktzahlen verwendet werden können. Die Gleitpunktregister werden durch die Nummern 0, 2, 4 oder 6 in einem R-Feld der Gleitpunktbefehle bestimmt. Jedes Gleitpunktregister ist 64 Bit lang und kann eine kurze oder eine lange Gleitpunktzahl aufnehmen. Kurze Gleitpunktzahlen werden in den linken 32 Bit eines Gleitpunktregisters gespeichert; in allen Gleitpunktbefehlen mit kurzen Gleitpunktoperanden werden die rechten 32 Bit ignoriert bzw. bleiben unverändert. Für erweiterte (128 Bit lange) Gleitpunktzahlen werden zwei aufeinanderfolgende Gleitpunktregister, also ein **Gleitpunktregister-Paar** verwendet, und zwar entweder das Gleitpunktregister-Paar aus den Gleitpunktregistern 0 und 2 oder das Gleitpunktregister-Paar aus den Gleitpunktregistern 4 und 6. In solchen Fällen ist dann im R-Feld für die erweiterten Gleitpunkt-Operanden die Nummer 0 bzw. 4 anzugeben.

2.3 Anzeige (Condition Code)

Die meisten Befehle "setzen die Anzeige", d.h. erzeugen während ihrer Ausführung einen Wert in einem internen Hardwareregister mit dem Namen "Anzeige". Die Anzeige (engl. Condition Code, abgekürzt CC) ist 2 Bit lang und kann auf die Werte 0_{10} oder 1_{10} oder 2_{10} oder 3_{10} gesetzt sein. Es gibt nur eine Anzeige; sie behält einen gesetzten Wert solange bis ein nachfolgender Befehl sie auf einen anderen Wert setzt.

Der häufigste Anwendungsfall für die Anzeige sind Vergleiche. Alle Vergleichsbefehle setzen die Anzeige gemäß dem von ihnen ermittelten Vergleichsergebnis, nämlich =0, wenn die verglichenen Operanden gleich sind, und =1 bzw. =2, wenn der erste Operand kleiner bzw. größer ist als der zweite. Nach einem Vergleichsbefehl kann der nachfolgende Befehl die gesetzte Anzeige abfragen und in Abhängigkeit von ihrem Wert geeignete Aktionen veranlassen.

Bei jedem Befehl ist beschrieben, ob und ggf. welche Werte er in der Anzeige setzt und welche Bedeutung diese Werte dann haben. Jedoch verwenden Befehle (Besonderheiten gibt es bei den Befehlen AL, ALR, SL, SPM und TM), die die Anzeige setzen, das folgende gemeinsame Schema zum Setzen auf einen der vier möglichen Werte:

Wert der Anzeige	Bedeutung
0	Das Resultat des Befehls ist =0. Nach Vergleichen bedeutet dies, daß der erste Operand gleich ist dem zweiten Operanden.
1	Das Resultat des Befehls ist <0. Nach Vergleichen bedeutet dies, daß der erste Operand kleiner ist als der zweite Operand.
2	Das Resultat des Befehls ist >0. Nach Vergleichen bedeutet dies, daß der erste Operand größer ist als der zweite Operand.
3	Bei der Befehlsausführung ist ein Überlauf eingetreten.

Um dem Leser das Merken dieser Tabelle zu erleichtern, ist in diesem Handbuch überall, wo ein Anzeigewert explizit auftritt, eine mnemotechnische Erläuterung beigefügt, die gleichzeitig einen Hinweis auf den ggf. einzusetzenden Abfragebefehl gibt: Wir sagen z.B. "die Anzeige wird auf 2 ~ High gesetzt" und weisen damit darauf hin, daß zur Abfrage dieses Anzeigewerts 2 der Befehl "Branch when High" verwendet werden kann.

2.4 Programmunterbrechungen

Wenn bei der Ausführung eines Befehls eine Ausnahmebedingung, z.B. ein falsch bestimmter Operand oder eine unzulässige Datenkonstellation entdeckt wird, erfolgt eine Programmunterbrechung. Wenn für einen solchen Fall im Anwenderprogramm keine besonderen Vorkehrungen getroffen sind, beendet das Betriebssystem BS2000 das Anwenderprogramm. Wenn allerdings im Anwenderprogramm (vor der ersten Programmunterbrechung) ein sog. STXIT-Prozeß definiert worden ist, aktiviert das BS2000 bei jeder Programmunterbrechung diesen Prozeß, so daß das Anwenderprogramm die Unterbrechung angemessen behandeln kann.

Jede mögliche Programmunterbrechung wird durch ihr **Unterbrechungsgewicht** identifiziert. Das Unterbrechungsgewicht ist eine zweistellige Sedezimalzahl, die bei vorhandenem STXIT-Prozeß diesem im Mehrzweckregister 3 mitgegeben wird und die bei fehlendem STXIT-Prozeß bei der Programmbeendigung auf SYSOUT ausgegeben wird.

Es gilt folgendes allgemeine Schema:

Art der Programm-unterbrechung	Unterbrechungsgewicht (sedezimal)	allgemeine Ursachen
Adreßumsetzungsfehler	48	Ein Operand enthält eine virtuelle Adresse, die nicht für das Anwenderprogramm bereitgestellt (allocated) ist. Auf diesen Operanden kann deshalb kein Lese- oder Schreibzugriff erfolgen.
Privilegierte Operation	54	Es wurde ein Befehl aufgerufen, der kein Assemblerbefehl, aber auch kein unzulässiger Befehl ist.
Falscher Operationscode	58	Es wurde ein unzulässiger Befehl aufgerufen.
Adreßfehler	5C	Die Nebenbedingung eines Befehls, z.B eine Ausrichtungs-Bedingung, ist nicht erfüllt.
Datenfehler	60	Ein Dezimaloperand enthält keine korrekte, gepackte Dezimalzahl oder zwei Dezimal-Operanden überlappen sich inkorrekt.
Exponenten-Überlauf	64	Die resultierende Charakteristik einer Gleitpunktoperation ist >127
Divisionsfehler	68	Bei einer Division ist der Divisor =0 oder der Quotient ist zu groß.
Signifikanz	6C	Die resultierende Mantisse einer Gleitpunktoperation ist =0.
Exponenten-Unterlauf	70	Die resultierende Charakteristik einer Gleitpunktoperation ist <0.
Dezimal-Überlauf	74	Das Resultat einer Dezimal-operation ist zu groß.
Festpunkt-Überlauf	78	Das Resultat einer Festpunkt-operation ist zu groß.

Die spezifische Ursache jeder Programmunterbrechung ist bei jedem einzelnen Befehl dargestellt; ausgenommen davon sind die Programmunterbrechungs-Arten *Privilegierte Operation* und *Falscher Operationscode*, die nicht befehlsspezifisch sind.

Bei einer Programmunterbrechung wird im allgemeinen Falle der verursachende Befehl nicht zu Ende ausgeführt und das Resultat des Befehls ist nicht richtig.

Maskierbare Programmunterbrechungen, Programmaske

Die Programmunterbrechungen wegen Festpunkt-Überlaufs, Dezimal-Überlaufs, Exponenten-Unterlaufs und Signifikanz sind **maskierbar**. Dies besagt, daß ein Anwenderprogramm bestimmen kann, ob in diesen Fällen eine Programmunterbrechung erfolgen soll. Die Maskierung geschieht in der **Programmaske**, einem 4 Bit langen internen Register der Zentraleinheit. Jedem der 4 Bit ist eine der genannten Programmunterbrechungen zugeordnet. Es bedeutet ein Bitwert 1, daß beim Auftreten der entsprechenden Ursache eine Programmunterbrechung erfolgt und ein Bitwert 0, daß eine solche nicht erfolgt. Das BS2000 besetzt zum Zeitpunkt des Starts eines Anwenderprogramms alle 4 Bit mit dem Wert 1 vor, so daß standardmäßig die vier Programmunterbrechungen erfolgen. Allerdings kann das Anwenderprogramm mit dem Befehl SPM die Vorbesetzung ändern und durch Setzen einzelner oder aller vier Bitwerte auf 0 die zugehörige Programmunterbrechung unterbinden.

Die Bit der Programmaske haben folgende Bedeutung:

Bit der Programmaske	Bedeutung
0	Festpunkt-Überlauf
1	Dezimal-Überlauf
2	Exponenten-Unterlauf
3	Signifikanz (Mantisse=0)

Hinweis

Der BS2000-Makro STXIT, mit dem STXIT-Prozesse zur Behandlung von Programmunterbrechungen definiert werden können, ist zusammen mit seinen Parametern in dem Handbuch "BS2000, Makroaufrufe an den Ablaufteil" [3] dargestellt.

2.5 Datentypen

Die Assemblerbefehle verwenden folgende Datentypen: Zeichen, Zeichenfeld, Binärzahl, Bitfeld, Dezimalzahl und Gleitpunktzahl. Von diesen sind die Datentypen Dezimalzahl und Gleitpunktzahl am Anfang der Kapitel 4 und 5 beschrieben, in denen die (sie verwendenden) Dezimalbefehle bzw. Gleitpunktbefehle erläutert sind, die anderen Datentypen werden nachfolgend dargestellt.
(Dem Sprachgebrauch folgend, wird der Datentyp selbst als Bezeichnung verwendet, wenn eigentlich eine Instanz des Datentyps gemeint ist: statt des umständlichen "Daten vom Datentyp X" wird einfach "X" geschrieben.)

2.5.1 Zeichen und Zeichenfelder

Der Datentyp **Zeichen** ist für Einzelzeichen vorgesehen, die z.B. von einer Tastatur kommen oder auf einen Drucker ausgegeben werden. Beispiele für solche Datenelemente sind die Buchstaben unseres Alphabets oder die Interpunktionszeichen in Texten.

Jedes Zeichen wird in einem Byte dargestellt. Die Abbildung eines Zeichens auf die 8 Bit eines Byte ist durch den **EBCDI-Code** bestimmt. Dieser Code bestimmt z.B., daß das Zeichen ’A’ binär durch die Codierung $(11000001)_2$, d.h. sedezimal durch $(C1)_{16}$ repräsentiert wird.
Eine vollständige EBCDIC-Tabelle findet sich im Anhang.

Der Datentyp **Zeichenfeld** ist für eine Menge zusammengehöriger Zeichen (= Daten vom Datentyp "Zeichen") vorgesehen, z.B. für ein Wort der Sprache oder auch einen gesamten Text. Ein Zeichenfeld wird in aufeinanderfolgenden Byte des Hauptspeichers dargestellt. Es wird gegenüber den Befehlen, die Zeichenfelder verarbeiten, durch zwei Angaben identifiziert: durch die Adresse des ersten (höchstwertigen) Byte und durch die "Länge", d.h. durch die Anzahl der Byte, die das Zeichenfeld umfaßt.

Vergleich von Zeichen und Zeichenfeldern

Der Vergleich von zwei Operanden vom Datentyp Zeichen oder Zeichenfeld erfolgt bitweise von links nach rechts; Zeichen und Zeichenfelder werden dabei als Bitfolgen behandelt. Bitstellen, die gleichweit vom Anfang ihrer Operanden liegen, heißen gegenüberliegend. Der Vergleich endet, wenn entweder *alle* gegenüberliegenden Bitstellen beider Operanden gleich sind oder wenn zwei gegenüberliegende Bitstellen verschieden sind. Im ersten Falle sind die Operanden "gleich". Im zweiten Falle sind sie "ungleich" und es ist derjenige Operand "kleiner", dessen zuletzt verglichene Bitstelle =0 ist; der andere Operand ist "größer".

2.5.2 Binärzahlen

Der Datentyp **Binärzahl** ist - neben den Datentypen Dezimalzahl und Gleitpunktzahl - für Daten vorgesehen, die arithmetisch behandelt, z.B. addiert werden sollen.

Binärzahlen sind Ganzzahlen zur Basis 2 mit einem angenommenen Binärpunkt rechts von der niedrigstwertigen Binärstelle. Jede Binärstelle einer Binärzahl wird in einem Bit dargestellt und zwar von links nach rechts in absteigender Wertigkeit.

Binärzahlen kommen in verschiedenen Längen vor. Die häufigste Länge beträgt 32 Bit und wird vor allem für Festpunktzahlen (siehe unten) verwendet. Es gibt jedoch auch Befehle für 16 Bit und 64 Bit lange Binärzahlen.

Ausrichtung von Binärzahlen

Zu ihrer Speicherung im Hauptspeicher benötigen Binärzahlen je nach ihrer Länge entweder 2 oder 4 oder 8 aufeinanderfolgende Byte. Die Adresse des ersten Byte muß ausgerichtet sein, d.h. durch 2 bzw 4 bzw. 8 ohne Rest teilbar sein. Man sagt auch, daß Binärzahlen an Halbwort- bzw. Wort- bzw. Doppelwortgrenzen ausgerichtet sein müssen.

Vorzeichen von Binärzahlen

Es gibt Binärzahlen mit oder ohne Vorzeichen. Binärzahlen mit Vorzeichen heißen Festpunktzahlen, Binärzahlen ohne Vorzeichen haben keinen eigenen Namen. Die Besonderheiten der beiden Arten sind im Folgenden erläutert.

Binärzahlen ohne Vorzeichen

Bei Binärzahlen ohne Vorzeichen werden alle Binärstellen zur Darstellung ihres Betrags verwendet; demzufolge sind bei arithmetischen und Vergleichs-Operationen alle Binärstellen an der Operation beteiligt. Die Begriffe "positiv" und "negativ" sind bei solchen Binärzahlen nicht relevant.

Der Wertebereich b Bit langer Binärzahlen ohne Vorzeichen reicht von 0 bis 2^b-1. Die (kleinste) Binärzahl mit dem Wert 0 wird durch lauter 0-Bit, die (größte) Binärzahl mit dem Wert 2^b-1 wird durch lauter 1-Bit dargestellt.

Binärzahlen mit Vorzeichen (Festpunktzahlen)

Bei Binärzahlen mit Vorzeichen - sie heißen **Festpunktzahlen** - wird die höchstwertige Binärstelle für das Vorzeichen verwendet; die Binärstellen rechts vom Vorzeichen repräsentieren den numerischen Wert der Festpunktzahl:

- Positive Festpunktzahlen werden durch ihren (Absolut)-Betrag dargestellt und haben ein 0-Bit an der höchstwertigen Binärstelle.
- Negative Festpunktzahlen werden durch das **Zweierkomplement** ihres (Absolut)-Betrags dargestellt und haben ein 1-Bit an der höchstwertigen Binärstelle.

Das Zweierkomplement einer Zahl ist definiert als die Differenz aus 0 und dem Betrag dieser Zahl, wobei ein etwaiger Übertrag über die höchstwertige Binärstelle ignoriert wird.

Für ausgewählte Werte aus dem Wertebereich einer 32 Bit langen Festpunktzahl ergeben sich folgende (binäre und sedezimale) Darstellungen:

Festpunktzahl	Binäre Darstellung	Sedezimale Darstellung
	<——32 Bit——>	
0	0000..........0000	00 00 00 00
+1	0000..........0001	00 00 00 01
+2	0000..........0010	00 00 00 02
.		
.		
+2147483647 = $+2^{31}-1$	0111..........1111	7F FF FF FF
-1	1111..........1111	FF FF FF FF
-2	1111..........1110	FF FF FF FE
.		
.		
-2147483648 = -2^{31}	1000..........0000	80 00 00 00

Festpunktzahlen haben folgende wesentliche Eigenschaften:

- der Wertebereich positiver, b Bit langer Festpunkzahlen reicht von 0 bis $2^{b-1}-1$, der Wertebereich negativer, b Bit langer Festpunktzahlen reicht von -1 bis -2^{b-1}. Für 32 Bit lange Festpunktzahlen liegt der Wertebereich demzufolge zwischen $-2^{31}=-2147483648$ und $+2^{31}-1=+2147483647$. Die kleinste negative Festpunktzahl hat also kein positives Pendant.
- Die Menge der Festpunktzahlen, die <0 sind, ist um Eins größer als die der Festpunktzahlen, die >0 sind.
- Es gibt keine negative Null.
- Die höchstwertige signifikante Binärstelle einer (positiven oder negativen) Festpunktzahl ist die höchstwertige Binärstelle, die ungleich dem Vorzeichenbit ist.

Hinweis

Alternative Methoden für die Bildung des Zweierkomplements einer Binärzahl sind:

1. Man invertiere alle Bitstellen der Binärzahl, addiere +1 auf die so gebildete Zahl und ignoriere einen etwaigen Übertrag über die höchstwertige Binärstelle.
2. Man invertiere alle Bitstellen links von der niedrigstwertigen Eins und lasse die niedrigstwertige Eins selbst sowie die rechts davon stehenden Binärstellen (sie sind alle =0) unverändert.

Vorzeichengerechte und logische Arithmetik von Binärzahlen

Es werden zwei Arten der Arithmetik von Binärzahlen unterschieden; die vorzeichengerechte (engl. signed) und die logische (engl. unsigned, logical) Binärzahlenarithmetik. Bei der vorzeichengerechten Binärzahlenarithmetik wird die höchstwertige Binärstelle jedes Operanden und des Resultats gesondert als Vorzeichen behandelt, während bei der logischen Binärzahlenarithmetik die höchstwertige Binärstelle genau wie die übrigen Binärstellen behandelt wird. Bei den einzelnen arithmetischen Operationen bestehen folgende Unterschiede.

Addition und Subtraktion von Binärzahlen

Die vorzeichengerechte Addition wird dadurch ausgeführt, daß alle Binärstellen beider Summanden, einschließlich der Vorzeichenstellen, addiert werden. Wenn einer der Summanden kürzer ist als der andere, wird er so behandelt, als sei er durch Binärstellen, die gleich sind dem Wert der Vorzeichenstelle, auf die Länge des längeren Summanden aufgefüllt.

Die logische Addition besteht ebenfalls in der Addition aller Binärstellen beider Summanden. Wenn jedoch ein Summand kürzer ist, wird er so behandelt, als sei er links durch Binärstellen mit dem Wert 0 auf die Länge des längeren Summanden aufgefüllt. Alle Adreßberechnungen werden mittels logischer Addition durchgeführt.

Die vorzeichengerechte wie die logische Subtraktion besteht in der vorzeichengerechten bzw. logischen Addition des Einerkomplements des zweiten Operanden und der Zahl 1 zum ersten Operanden. (Das Einerkomplement einer Binärzahl entsteht durch Invertierung aller Bitstellen der Zahl).

Der Unterschied zwischen vorzeichengerechter und logischer Addition bzw. Subtraktion besteht in der Interpretation des Resultats:

– Bei einer logischen Addition bzw. Subtraktion wird das Resultat als vorzeichenlose Binärzahl aufgefaßt; in der Anzeige wird dargestellt, ob das Resultat =0 oder ≠0 ist und ob ein Überlauf aus der höchstwertigen Binärstelle, d.h. ein Übertrag über die Bitstelle 0 hinaus stattgefunden hat oder nicht.

– Bei einer vorzeichengerechten Addition bzw. Subtraktion wird das Resultat als Binärzahl mit Vorzeichen (Festpunktzahl) aufgefaßt; in der Anzeige wird dargestellt, ob das Resultat =0, <0 oder >0 ist oder ob ein Festpunkt-Überlauf eingetreten ist. Festpunkt-Überlauf ist eingetreten, wenn ein etwaiger Binärstellenüberlauf *in* die Vorzeichenstelle des Resultats ungleich ist dem Binärstellenüberlauf *aus* der Vorzeichenstelle. Arithmetisch gesehen bedeutet dies im Falle 32 Bit langer Festpunktzahlen, daß das Resultat größer als $+2^{31}-1$ oder kleiner als -2^{31} ist. Bei Festpunkt-Überlauf wird die Anzeige auf 3~Overflow gesetzt und es erfolgt außerdem eine Programmunterbrechung, wenn in der Programmaske das Bit für Festpunkt-Überlauf =1 gesetzt ist (BS2000-Standard).

Links- oder Rechts-Verschiebung von Binärzahlen

Die vorzeichengerechte Verschiebung einer Binärzahl behandelt die Vorzeichenstelle gesondert, die logische Verschiebung dagegen nicht.

Bei vorzeichengerechter Verschiebung bleibt die Vorzeichenstelle stets unverändert und nur die Binärstellen ab Bitstelle 1 werden verschoben; bei Verschiebung nach rechts werden links freiwerdende Binärstellen mit dem Bitwert der Vorzeichenstelle besetzt, bei Verschiebung nach links werden rechts freiwerdende Binärstellen mit 0 besetzt. Wenn bei einer Linksverschiebung signifikante, d.h. von der Vorzeichenstelle verschiedene Binärstellen links über die Bitstelle 1 hinaus geschoben werden, tritt Festpunkt-Überlauf ein; die Anzeige wird dann auf 3~Overflow gesetzt und es erfolgt außerdem eine Programmunterbrechung, wenn das Bit für Festpunkt-Überlauf in der Programmmaske =1 gesetzt ist (BS2000-Standard).

Bei logischer Links- oder Rechtsverschiebung einer Binärzahl werden alle Binärstellen, auch die Vorzeichenstelle, verschoben. Wenn nach rechts verschoben wird, werden links frei werdende Binärstellen mit 0 aufgefüllt, wenn nach links verschoben wird, werden rechts freiwerdende Binärstellen mit 0 aufgefüllt. Die Anzeige wird bei logischer Verschiebung nicht verändert.

Vergleich von Binärzahlen

Der vorzeichengerechte Vergleich von zwei Binärzahlen wird so durchgeführt, als werde eine vorzeichengerechte Subtraktion ausgeführt, bei der das Resultat nicht gespeichert wird. Die Anzeige wird auf 0~Equal, bzw. 1~Low bzw. 2~High gesetzt, je nachdem ob der erste Operand gleich oder kleiner oder größer ist als der zweite. Festpunkt-Überlauf kann nicht auftreten.

Der logische Vergleich von zwei Binärzahlen besteht aus einem bitweisen Vergleich der beiden Zahlen von links nach rechts. Der Vergleich wird beendet, wenn entweder die beiden Operanden abgearbeitet sind oder zwei gegenüberliegende Bit verschieden sind. Die Anzeige wird bei Gleichheit der Operanden auf 0~Equal gesetzt und bei Ungleichheit entweder auf 1~Low oder 2~High, je nachdem ob die zuletzt verglichene Bitstelle des ersten Operanden =0 oder =1 war.

2.5.3 Bitfeld

Der Datentyp **Bitfeld** ist ein Datentyp für Folgen von 1 Bit-Werten. Jeder Bit-Wert wird in einer Bitstelle dargestellt. Die einzelnen Bitstellen werden in den Befehlen für Bitfelder unabhängig von den übrigen Bitstellen behandelt. Bitfelder beginnen oder enden an Bytegrenzen. Die Bitstellen eines Bitfeldes werden gewöhnlich von links nach rechts ab 0 numeriert, aber diese Konvention ist für die Zentraleinheit unerheblich.

Ein häufig vorkommender Anwendungsfall des Datentyps Bitfeld sind **Masken**. Masken dienen der Auswahl einzelner Bit oder Byte eines Registers oder des Hauptspeichers oder der Folgeaktionenen eines Befehls. Ihre konkrete Bedeutung und Wirkung ist bei den einzelnen Befehlen, die Masken verwenden, beschrieben.

2.6 Befehlsaufbau

Jeder Befehl besteht aus zwei Teilen:

1. aus dem sog. Operationscode, der die Wirkung des Befehls bestimmt, und
2. aus der Angabe seiner Operanden.

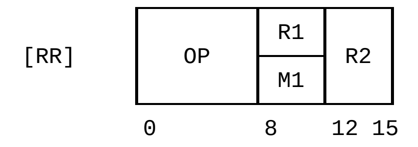
Alle Befehle müssen im Hauptspeicher an Halbwortgrenzen ausgerichtet sein. Je nach Befehlstyp sind Befehle entweder 2, 4 oder 6 Byte lang.

Befehlstypen

Es gibt die folgenden generellen Befehlstypen:

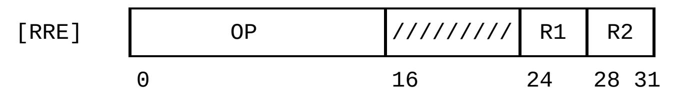
1. Befehlstyp RR

Für Befehle mit zwei Register-Operanden oder einem Register-Operanden und einer Maske.



2. Befehlstyp RRE

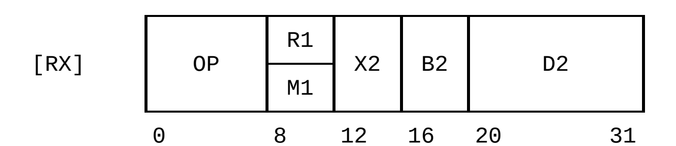
Für Befehle mit erweitertem Operationscode und zwei Register-Operanden.



Die Bitstellen 16 bis 23 werden bei Befehlen dieses Befehlstyps ignoriert.

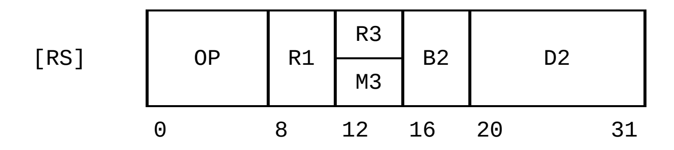
3. Befehlstyp RX

Für Befehle mit einem Register-Operanden oder einer Maske und einem index-adressierten Hauptspeicher-Operanden.



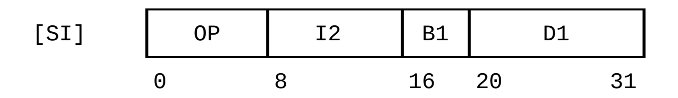
4. Befehlstyp RS

Für Befehle mit zwei Register-Operanden und einem Hauptspeicher-Operanden oder einem Register-Operanden und einem Hauptspeicher-Operanden und einer Maske.



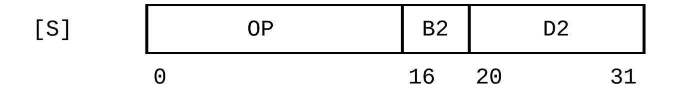
5. Befehlstyp SI

Für Befehle mit einem Hauptspeicher-Operanden und einem Direktoperanden.



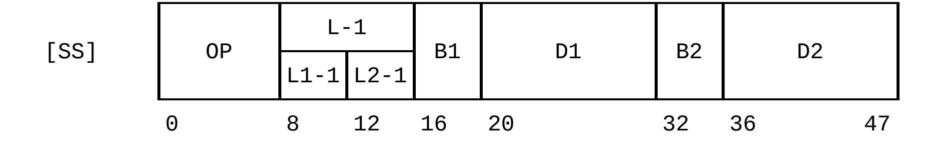
6. Befehlstyp S

Für Befehle mit erweitertem Operationscode und einem Hauptspeicher-Operanden.



7. Befehlstyp SS

Für Befehle mit zwei Speicher-Operanden mit gleicher oder verschiedener Operanden-Länge. Die L-, L1- und L2-Felder enthalten die um Eins verminderte Länge.



Legende

In den vorhergehend beschriebenen Befehlstypen bedeuten:

Bezeichnung	Länge	Bedeutung
OP	8/16 Bit	Operationscode
R1,R2,R3	4 Bit	Mehrzweck-, Zugriffs- oder Gleitpunktregister
M1,M3	4 Bit	Maske
B1,B2	4 Bit	Basisregister
X1,X2	4 Bit	Indexregister
D1,D2	12 Bit	Distanzadresse
I2	8 Bit	Direktoperand
L-1	8 Bit	Operanden-Länge minus 1
L1-1, L2-1	4 Bit	Operanden-Länge minus 1

Der Operationscode (OP) ist eine befehlsspezifische zwei- oder vierstellige sedezimale Zahl. Sie ist bei jedem Befehl genannt und in Assemblerschreibweise dargestellt, z.B. beim Befehl MVN durch X’D1’. Die ersten beiden Bitstellen des Operationscodes bestimmen die Länge des Befehls wie folgt:

Bitstellen 0 und 1 des Operationscodes	Befehlstyp(en)	Befehlslänge
00	RR	2 Byte
01	RX	4 Byte
10	RRE/RS/S/SI	4 Byte
11	SS	6 Byte

Befehlsoperanden

Die an einer Befehlsausführung beteiligten Operanden werden rechts vom Operationscode in befehlstyp-spezifischen Feldern bestimmt. Je nach Befehlstyp sind bei einem Befehl bis zu drei Operanden beteiligt. Sie werden in diesem Handbuch Operand1, Operand2 und Operand3 genannt. Die Bestimmungsgrößen dieser Operanden im Befehl sind durch die Suffixe "1", "2" und "3" voneinander unterschieden.

Befehlsoperanden können entweder direkt im Befehl oder getrennt vom Befehl entweder in einem (Mehrzweck- oder Gleitpunkt-)Register oder im Hauptspeicher gespeichert sein. Davon abhängig spricht man von Direktoperanden, Register-Operanden oder Hauptspeicher-Operanden.

Ein **Direktoperand** wird im Befehl in einem M- oder I-Feld als Bitfeld dargestellt.

Ein **Register-Operand** wird durch die Angabe der entsprechenden, 4 Bit langen Registernummer in einem R-Feld bestimmt. Durch den Befehl ist dabei festgelegt, ob es sich um ein Mehrzweckregister, ein Zugriffsregister oder ein Gleitpunktregister handelt.

Ein **Hauptspeicher-Operand** wird durch die Adresse (seines höchstwertigen Byte) und durch seine Länge (in Byte) bestimmt. Seine Adresse ergibt sich entweder aus einer Adreßberechnung oder wird als fertige Adresse einem Mehrzweckregister entnommen. Im Falle einer Adreßberechnung besteht diese aus der logischen Addition einer Distanzadresse (direkt bestimmt in einem D-Feld) und dem Inhalt von einem oder zwei Mehrzweckregistern, deren Registernummer in einem B- und einem X-Feld bestimmt sind (siehe oben, Adreßberechnung). Der Assembler [1] errechnet bei sog. symbolischen Adressen von Befehlen und Daten selbständig die Adreßkomponenten B und D.

Die Länge von Hauptspeicher-Operanden wird entweder implizit *durch* den Befehl oder explizit *im* Befehl bestimmt. Bei impliziter Länge ist diese bei der Befehlsbeschreibung dargestellt, bei expliziter Angabe (in Befehlen des SS-Typs) wird die Operanden-Länge durch ihren Wert *minus 1* im Befehl bestimmt, und zwar in einem Feld, das in der obigen Darstellung durch "L-1" oder "L1-1" oder "L2-1" bezeichnet ist. (Der Assembler [1] generiert die Inhalte dieser Längenfelder automatisch, indem er die Operanden-Länge um 1 reduziert).

3 Allgemeine Befehle

Add

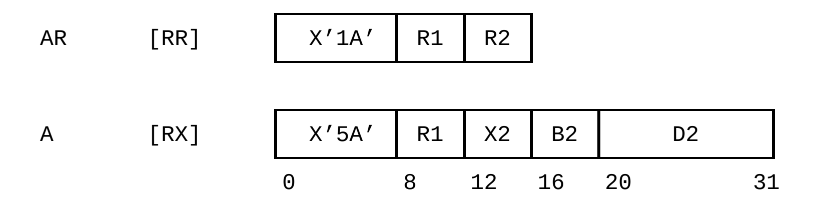
Funktion

Die Befehle AR und A addieren zwei 32 Bit lange Festpunktzahlen vorzeichengerecht. Die Anzeige wird gemäß dem Wert der Summe gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	AR A	R1,R2 R1,D2(X2,B2)	 D2(X2,B2): Wortgrenze

Maschinenformate



Beschreibung

Durch den Befehl AR wird der Inhalt des Mehrzweckregisters R2, durch den Befehl A wird das mit D2(X2,B2) adressierte Wort des Hauptspeichers vorzeichengerecht zum Inhalt des Mehrzweckregisters R1 addiert. Beide Operanden werden als 32 Bit lange Binärzahlen mit Vorzeichen (Festpunktzahlen) behandelt. Die Summe ist ebenfalls eine 32 Bit lange Festpunktzahl und ersetzt den ursprünglichen Inhalt des Mehrzweckregisters R1.

Festpunkt-Überlauf entsteht, wenn die Summe größer als 2^{31}-1 bzw. kleiner als -2^{31} wird. In diesem Fall ist das Ergebnis in R1 um 2^{32} zu klein bzw. zu groß; die Anzeige wird dann auf 3~Overflow gesetzt und es erfolgt eine Programmunterbrechung, wenn in der Programmaske das Bit für Festpunkt-Überlauf =1 ist (BS2000-Standard).

Anzeige

0~Zero	Summe = 0
1~Minus	Summe < 0
2~Plus	Summe > 0
3~Overflow	Festpunkt-Überlauf

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	A: Lesezugriff auf Operand2 unmöglich
Adreßfehler	X'5C'	A: D2(X2,B2) keine Wortgrenze.
Festpunkt-Überlauf	X'78'	Summe > $+2^{31}$-1 oder < -2^{31}

Programmierhinweise

Festpunkt-Überlauf entsteht dann, wenn ein Binärstellenüberlauf in die Vorzeichenstelle ungleich ist dem Binärstellenüberlauf aus der Vorzeichenstelle. Im Register R1 hat dann das Resultat ein falsches Vorzeichen.

Beispiele

Name	Operation	Operanden	
	.		
Beispiel1	L	15,=F'-2147483647'	Reg 15: X'80000001' = -2^{31}+1
	A	15,=F'-1'	Reg 15: X'80000000' = -2^{31}
*			Anzeige: 1~Minus
	.		
Beispiel2	LM	15,0,=F'2147483647'	Reg 15: X'7FFFFFFF' = $+2^{31}$-1
*	LA	0,1	Reg 0 : 1
	AR	15,0	Reg 15: X'80000000' = -2^{31}
*			Anzeige: 3~Overflow und
*			ggf. Programmunterbrechung
*			wegen Festpunkt-Überlaufs
	.		

Die Anzeige 3~Overflow in Beispiel2 weist darauf hin, daß das Resultat arithmetisch nicht korrekt ist.

Add Halfword

Funktion

Der Befehl AH addiert eine 16 Bit lange Festpunktzahl zu einer 32 Bit langen Festpunktzahl vorzeichengerecht.
Die Anzeige wird gemäß dem Wert der Summe gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	AH	R1,D2(X2,B2)	D2(X2,B2): Halbwortgrenze

Maschinenformat

AH [RX]

X'4A'	R1	X2	B2	D2
0	8	12	16	20 ... 31

Beschreibung

Das durch D2(X2,B2) adressierte Halbwort im Hauptspeicher wird vorzeichengerecht zum Inhalt des Mehrzweckregisters R1 addiert. Der Register-Operand wird als 32 Bit lange, der Halbwort-Operand als 16 Bit lange Festpunktzahl behandelt, beide mit Vorzeichen. Die Summe ist eine 32 Bit lange Festpunktzahl mit Vorzeichen und ersetzt den ursprünglichen Inhalt des Mehrzweckregisters R1.

Festpunkt-Überlauf entsteht, wenn die Summe größer als 2^{31}-1 oder kleiner als -2^{31} wird. In diesem Fall ist das Ergebnis in R1 um 2^{32} zu klein bzw. zu groß; die Anzeige wird dann auf 3~Overflow gesetzt und es erfolgt eine Programmunterbrechung, wenn in der Programmaske das Bit für Festpunkt-Überlauf =1 ist (BS2000-Standard).

Anzeige

0~Zero	Summe = 0
1~Minus	Summe < 0
2~Plus	Summe > 0
3~Overflow	Festpunkt-Überlauf

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	D2(X2,B2) keine Halbwortgrenze.
Festpunkt-Überlauf	X’78’	Summe > $+2^{31}$-1 oder < -2^{31}

Programmierhinweise

Festpunkt-Überlauf entsteht dann, wenn ein Binärstellenüberlauf in die Vorzeichenstelle ungleich ist dem Binärstellenüberlauf aus der Vorzeichenstelle. Im Register R1 hat dann das Resultat ein falsches Vorzeichen.

Beispiel

Name	Operation	Operanden	
	.		
	L	13,=F’16999999’	
	AH	13,=H’+1’	Register 13: F’17000000’
*			Anzeige 2~Plus
	.		

In vielen Programmen findet man solche einfachen Register-Inkrementierungen mit dem Befehl LA 13,1(13) ausgeführt und nicht wie hier mit dem Befehl AH 13,=H’+1’. Ein solcher LA-Befehl wäre jedoch in obigem Beispiel nicht sinnvoll: Wenn nämlich das Programm im 24-Bit-Adressierungsmodus abläuft, liefert der Befehl LA nicht das (vermutlich gewünschte) Ergebnis 17000000, sondern das ganz andere Resultat 222784, d.h. 17000000 modulo 2^{24}. Nur wenn das Programm im 31-Bit-Adressierungsmodus ausgeführt wird, ergäbe sich nach dem LA der Wert 17000000. Adressen, wie sie der Befehl LA erzeugt, sind eben etwas anderes als Festpunktzahlen.

Add Logical

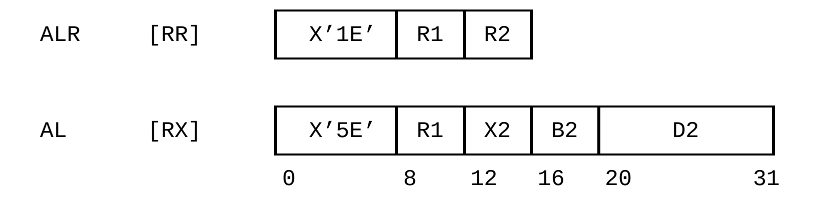
Funktion

Die Befehle ALR und AL addieren zwei 32 Bit lange Binärzahlen logisch.
Die Anzeige wird gemäß dem Wert der Summe gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	ALR AL	R1,R2 R1,D2(X2,B2)	 D2(X2,B2): Wortgrenze

Maschinenformate



Beschreibung

Durch den Befehl ALR wird der Inhalt des Mehrzweckregisters R2, durch den Befehl AL wird das mit D2(X2,B2) adressierte Wort des Hauptspeichers logisch zum Inhalt des Mehrzweckregisters R1 addiert.

Die Summe ist eine 32 Bit lange Binärzahl ohne Vorzeichen und ersetzt den ursprünglichen Inhalt des Mehrzweckregisters R1.

Alle 32 Bit beider Operanden sind an der Addition beteiligt. Ein Übertrag über die Bitstelle 0 hinaus wird in der Anzeige dargestellt.

Anzeige

0~Zero	Summe = 0, kein Übertrag
1~Minus	Summe ≠ 0, kein Übertrag
2~Plus	Summe = 0, Übertrag
3~Overflow	Summe ≠ 0, Übertrag

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Adreßfehler	X’48’ X’5C’	AL: Lesezugriff auf Operand2 unmöglich AL: D2(X2,B2) keine Wortgrenze.

Programmierhinweise

- Die Anzeige 0~Zero wird nur dann gesetzt, wenn beide Operanden =0 sind.
- Der Befehl AL kann Anwendung finden bei der vorzeichengerechten Addition von Festpunktzahlen, die länger sind als 32 Bit. Dabei verwendet man AL-Befehle zur Addition der niedrigstwertigen Wortpaare und benutzt den Befehl A zur Addition des höchstwertigen Wortpaars; bei einem Übertrag bei der Addition eines niedrigstwertigen Wortpaars muß dann die Zahl +1 auf die Summe des nächsthöheren Wortpaars addiert werden (siehe Beispiel2).

Beispiele

Name	Operation	Operanden	
	.		
Beispiel1	L	15,=F’-1’	
	AL	15,=F’1’	Register 15: 0, Anzeige: 2~Plus
	.		
Beispiel2	LM	0,1,FPNO1	Addition von zwei 64 Bit langen
LOWADD	AL	1,FPNO2+4	Festpunktzahlen
	BC	12,HIGHADD	
	AH	0,=H’1’	Überlauf im rechten Teil
HIGHADD	A	0,FPNO2	
	.		

Das Beispiel2 zeigt die vorzeichengerechte Addition von zwei 64 Bit langen Festpunktzahlen FPNO1 und FPNO2: Das niederwertige Wortpaar wird mittels AL und das höherwertige Wortpaar wird mittels A addiert. Im Falle von Übertrag bei der Addition des niederwertigen Wortpaars muß zur Summe des höherwertigen Wortpaars noch +1 addiert werden. Das Ergebnis steht im Beispielsfalle in den Mehrzweckregistern 0 und 1.

Branch and Link

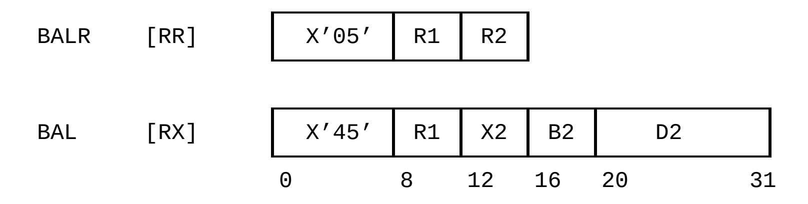
Funktion

Die Befehle BALR und BAL speichern die Befehlsfolgeadresse in ein angegebenes Mehrzweckregister und verzweigen dann an eine angegebene Adresse.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	BALR BAL	R1,R2 R1,D2(X2,B2)	

Maschinenformate



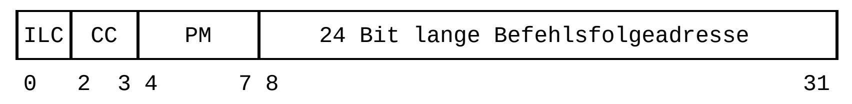
Beschreibung

Die Befehle BALR und BAL speichern zunächst die Befehlsfolgeadresse in das Mehrzweckregister R1.
Das Format der gespeicherten Adresse hängt vom Adressierungsmodus ab:

– 24-Bit-Adressierungsmodus:

ILC	CC	PM	24 Bit lange Befehlsfolgeadresse
0	2 3	4 7	8 31

– 31-Bit-Adressierungsmodus:



Darin bedeuten:

- ILC Instruction Length Code; 01_2 bei BALR und 10_2 bei BAL.
- CC Condition Code; momentaner Wert der Anzeige: 0, 1, 2 oder 3
- PM momentaner Wert der Programmaske:
  BS2000-Standardbesetzung ist F_{16}, aber dieser Wert kann durch das Anwenderprogramm mittels SPM geändert worden sein.
- A Adressing Mode; =1 beim 31-Bit-Adressierungsmodus.

Nach der Speicherung der Befehlsfolgeadresse verzweigt der Befehl BALR an die im Mehrzweckregister R2 enthaltene Adresse und der Befehl BAL an die Adresse D2(X2,B2). (Diese Adresse ist je nach Adressierungsmodus 24 Bit oder 31 Bit lang). Bei BALR erfolgt die Verzweigung nicht, wenn das R2-Feld =0 ist. In jedem Fall bleibt der momentane Adressierungsmodus bestehen.

Die Sprungadresse wird ermittelt, bevor das Mehrzweckregister R1 verändert wird.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine beim Befehl selbst.
Wenn allerdings die Zieladresse keine Halbwortadresse ist oder auf sie nicht zugegriffen werden kann, erfolgt eine entsprechende Programmunterbrechung (mit dem Gewicht $5C_{16}$ bzw. 48_{16}) an der Zieladresse.

Programmierhinweise

- Zur Rückkehr aus einem mit BALR oder BAL aufgerufenen Unterprogramm sollte der Befehl BCR (oder der Befehl BC) verwendet werden; die Verwendung des Befehls BSM für diesen Zweck ist nicht möglich, wenn das Unterprogramm mittels BAL gerufen wurde und der Aufruf im 24-Bit-Adressierungsmodus erfolgt ist.
- Die ’Befehlsfolgeadresse’ ist bei direkter Ausführung eines BALR-Befehls dessen um 2 erhöhte Befehlsadresse und bei direkter Ausführung eines BAL-Befehls dessen um 4 erhöhte Befehlsadresse; wenn allerdings ein Befehl BALR oder BAL mittels des Befehls EX ausgeführt wird, ist die Befehlsfolgeadresse die um 4 erhöhte Befehlsadresse dieses EX.

– Man beachte, daß die Befehle BALR und BAL im 24-Bit-Adressierungsmodus in das Mehrzweckregister R1 keine "reinen" 24 Bit Adressen erzeugen, sondern das obere Byte mit adressfremden Informationen versorgen. Dies macht diese Befehle ungeeignet zur Verwendung in Programmen, die sowohl im 24-Bit-Adressierungsmodus als auch im 31-Bit-Adressierungsmodus ablaufen sollen. In solchen Programmen sollten statt BAL oder BALR die Befehle BAS oder BASR verwendet werden. (Die von den Befehlen BAL und BALR gelieferten Werte der Anzeige (CC) oder der Programmaske (P'Mask) können in portabler Weise mit dem Befehl IPM gewonnen werden).

– Man beachte folgenden Unterschied zwischen BALR und BAL: bei BALR wird die Sprungadresse durch den *Inhalt*, bei BAL jedoch durch die *Adresse* des zweiten Operanden bestimmt.

Branch and Save

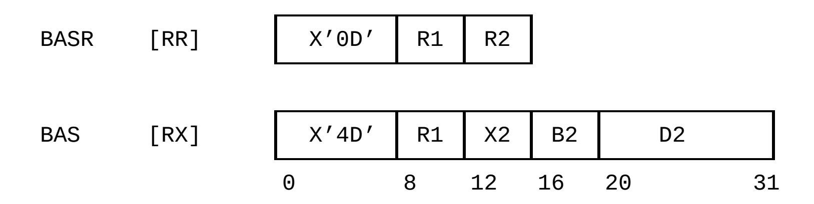
Funktion

Die Befehle BASR und BAS speichern den momentanen Adressierungsmodus und die Befehlsfolgeadresse in ein Mehrzweckregister und verzweigen unter Beibehaltung des momentanen Adressierungsmodus an eine angegebene Adresse.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	BASR BAS	R1,R2 R1,D2(X2,B2)	

Maschinenformate



Beschreibung

Der momentane Adressierungsmodus wird in die Bitstelle 0 des Mehrzweckregisters R1 und die Befehlsfolgeadresse wird in die Bitstellen 1 bis 31 des Mehrzweckregisters R1 gespeichert. Der 24-Bit-Adressierungsmodus wird durch den Wert 0, der 31-Bit-Adressierungsmodus wird durch den Wert 1 an der Bitstelle 0 dargestellt.

Anschließend wird unter Beibehaltung des momentanen Adressierungsmodus beim Befehl BASR an die im Mehrzweckregister R2 enthaltene Adresse und beim Befehl BAS an die Adresse D2(X2,B2) verzweigt. Diese Adresse ist je nach Adressierungsmodus 24 Bit oder 31 Bit lang. Wenn beim Befehl BASR das R2-Feld =0 ist, wird nicht verzweigt, sondern mit dem nächsten Befehl fortgefahren.

Die Sprungadresse wird ermittelt, bevor das Mehrzweckregister R1 verändert wird.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine beim Befehl selbst.
Wenn allerdings die Zieladresse keine Halbwortadresse ist oder auf sie nicht zugegriffen werden kann, erfolgt eine entsprechende Programmunterbrechung (mit dem Gewicht $5C_{16}$ bzw. 48_{16}) an der Zieladresse.

Programmierhinweise

- Die Befehle BASR und BAS dienen dem Aufruf eines Unterprogramms, das im gleichen Adressierungsmodus abläuft wie das aufrufende Programm.
- Der Aufruf von BASR mit einem R2-Feld =0 bewirkt, daß nur der momentane Adressierungsmodus und die Befehlsfolgeadresse ins Mehrzweckregister R1 gespeichert werden, aber nicht verzweigt wird. Dies kann man z.B. zur Gewinnung einer "Basisadresse" benutzen, etwa in der Anweisungsfolge

  ```
  BASR   3,0
  USING  *,3
  ```
- Zur Rückkehr aus einem mit BASR oder BAS aufgerufenen Unterprogramm wird normalerweise der Befehl BCR (oder der Befehl BC) verwendet, jedoch kann auf Zentraleinheiten, die über den 31-Bit-Adressierungsmodus verfügen, dafür auch der Befehl BSM verwendet werden.
- Die ’Befehlsfolgeadresse’ ist bei direkter Ausführung eines BASR-Befehls dessen um 2 erhöhte Befehlsadresse und bei direkter Ausführung eines BAS-Befehls dessen um 4 erhöhte Befehlsadresse; wenn allerdings ein BASR- oder BAS-Befehl mittels des Befehls EX ausgeführt wird, ist die Befehlsfolgeadresse die um 4 erhöhte Adresse des EX.
- Die Befehle BASR und BAS leisten das Gleiche wie die Befehle BALR und BAL. BASR und BAS erzeugen jedoch reine Befehlsfolgeadressen auch im 24-Bit-Adressierungsmodus und besetzen das höchstwertige Byte im Mehrzweckregister R1 nicht mit adressfremden Informationen (z.B. ILC), die zu Problemen im 31-Bit-Adressierungsmodus führen können. Allerdings sind die Befehle BASR und BAS nicht im Befehlsvorrat älterer Zentraleinheiten verfügbar.
- Man beachte folgenden Unterschied zwischen BASR und BAS: bei BASR wird die Sprungadresse durch den *Inhalt*, bei BAS jedoch durch die *Adresse* des zweiten Operanden bestimmt.

Beispiel

Zum Unterprogrammsprung aus einem Programmteil A in einen Programmteil B, die beide im selben Adressierungsmodus ablaufen (24 Bit oder 31 Bit), aber in verschiedenen Übersetzungseinheiten assembliert sind, kann der Befehl BASR so eingesetzt werden:

```
Name      Operation  Operanden          Name      Operation  Operanden
*
A         CSECT               ------->  B         CSECT
A         AMODE      31       |         B         AMODE      31
          .                   |                   .
          L          15,=V(B) |                   .
          BASR       14,15 ---                    .
          .               <---                    .
          .                   |                   .
          .                   ------------------  BR         14
*         .                                       .
```

Branch and Save and Set Mode

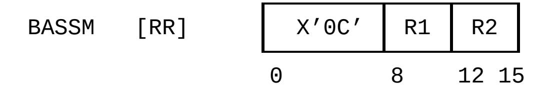
Funktion

Der Befehl BASSM speichert den momentanen Adressierungsmodus und die Befehlsfolgeadresse in ein Mehrzweckregister, setzt danach einen angegebenen Adressierungsmodus und verzweigt in diesem Modus an eine angegebene Adresse.
Die Anzeige wird nicht verändert.
Der Befehl BASSM ist nur im Befehlsvorrat der Zentraleinheiten verfügbar, die über den 31-Bit-Adressierungsmodus verfügen.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	BASSM	R1,R2	

Maschinenformat



Beschreibung

Der momentane Adressierungsmodus wird in die Bitstelle 0 des Mehrzweckregisters R1 und die Befehlsfolgeadresse wird in die Bitstellen 1 bis 31 des Mehrzweckregisters R1 gespeichert.

Wenn das R2-Feld $\neq$ 0 ist, wird anschließend der Adressierungsmodus gesetzt, der sich aus dem Bit 0 des Mehrzweckregisters R2 ergibt und es wird in diesem Modus an die in den Bitstellen 1 bis 31 von R2 enthaltene Adresse verzweigt. Diese Adresse ist je nach Adressierungsmodus 24 Bit oder 31 Bit lang. Wenn das R2-Feld =0 ist, wird nicht verzweigt, sondern im momentanen Adressierungsmodus beim nächsten Befehl fortgefahren.

In beiden Mehrzweckregistern R1 und R2 bedeutet ein Wert 0 an der Bitstelle 0 den 24-Bit-Adressierungsmodus und ein Wert 1 den 31-Bit-Adressierungsmodus.

Die Sprungadresse wird ermittelt, bevor das Mehrzweckregister R1 verändert wird.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine beim Befehl selbst.
Wenn allerdings die Zieladresse keine Halbwortadresse ist oder auf sie nicht zugegriffen werden kann, erfolgt eine entsprechende Programmunterbrechung (mit dem Gewicht $5C_{16}$ bzw. 48_{16}) an der Zieladresse.

Programmierhinweise

- Der Befehl BASSM dient dem Aufruf eines Unterprogramms, das im gleichen oder in einem anderen Adressierungsmodus abläuft als das aufrufende Programm.
- Die 'Befehlsfolgeadresse' ist bei direkter Ausführung eines BASSM-Befehls dessen um 2 erhöhte Befehlsadresse; wenn allerdings der Befehl BASSM durch den Befehl EX ausgeführt wird, ist die Befehlsfolgeadresse die 4 erhöhte Adresse dieses EX.
- Der Befehl BASSM ist nur auf Zentraleinheiten verfügbar, die über den 31-Bit-Adressierungsmodus verfügen. Um ein Assemblerprogramm unabhängig von der Zentraleinheit zu machen, auf der es abläuft, gibt es im Vorrat der Makroaufrufe des BS2000 (V9.0 und Nachfolger) den Makro ##BASSM, mit dem der Befehl BASSM "eingeschalt" wird. Der Makroaufruf ##BASSM generiert auf Anlagen mit 31-Bit-Adressierungsmodus den Befehl BASSM und auf Anlagen, die nur über den 24-Bit-Adressierungsmodus verfügen, den Befehl BALR.
  Eine vollständige Darstellung der Adressierungsmodi und der verschiedenen Programmverknüpfungsformen enthält das Handbuch "Einführung in die XS-Programmierung (für ASSEMBLER-Programmierer)" [2].
- Zur Rückkehr aus einem mit BASSM aufgerufenen Unterprogramm sollte der Befehl BSM verwendet werden, um sicherzustellen, daß anschließend wieder der Adressierungsmodus des aufrufenden Programms herrscht.

Beispiel

Zum Unterprogrammsprung aus einem Programmteil A, der im 24-Bit-Adressierungsmodus abläuft, in einen Programmteil B, der im 31-Bit-Adressierungsmodus abläuft, kann der Befehl BASSM so eingesetzt werden:

Name	Operation	Operanden
	.	
A	CSECT	
A	AMODE	24
	.	
	L	15,=V(B)
	O	15,AM31
	BASSM	14,15
	.	
	.	
	.	
	DS	0F
AM31	DC	X'80000000'
	.	

Name	Operation	Operanden
	.	
B	CSECT	
B	AMODE	31
	.	
	.	
	.	
	.	
	.	
	.	
	BSM	0,14
	.	
	.	
	.	

Branch on Condition

Funktion

Die Befehle BCR und BC bewirken in Abhängigkeit vom momentanen Wert der Anzeige eine Verzweigung an eine angegebene Adresse.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	BCR BC	M1,R2 M1,D2(X2,B2)	B'0000' ≤ M1 ≤ B'1111' B'0000' ≤ M1 ≤ B'1111'

Maschinenformate

BCR [RR]

X'07'	M1	R2

BC [RX]

X'47'	M1	X2	B2	D2
0	8	12	16	20 31

Beschreibung

Die "Maske" M1 ist ein 4 Bit langes Feld. Dessen vier Bitstellen entsprechen von links nach rechts den vier möglichen Werten der Anzeige wie folgt:

Wert der Anzeige	Bitstelle der Maske M1	Wert der Maske M1
0	0	8
1	1	4
2	2	2
3	3	1

Wenn zum Zeitpunkt der Befehlsausführung die Anzeige den Wert i hat und die Bitstelle i der Maske M1 =1 ist, verzweigt der Befehl BCR an die im Mehrzweckregister R2 enthaltene Adresse und der Befehl BC an die Adresse D2(X2,B2). Diese Adresse ist je nach Adressierungsmodus 24 Bit oder 31 Bit lang. Wenn die Bitstelle i der Maske M1 =0 ist, wird nicht verzweigt, sondern mit dem nachfolgenden Befehl fortgefahren; ebenso wird nicht verzweigt, wenn beim Befehl BCR das R2-Feld =0 ist.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine beim Befehl selbst.
Wenn allerdings tatsächlich verzweigt wird und die Zieladresse keine Halbwortadresse ist oder auf sie nicht zugegriffen werden kann, erfolgt eine entsprechende Programmunterbrechung (mit dem Gewicht $5C_{16}$ bzw. 48_{16}) an der Zieladresse.

Programmierhinweise

– In der Maske M1 darf mehr als eine oder keine Bitstelle =1 gesetzt sein; verzweigt wird bei jedem Anzeige-Wert, dessen entsprechendes Maskenbit =1 ist. Demzufolge wird bei einer Maske M1 aus lauter Einsen in jedem Falle und bei einer Maske aus lauter Nullen in keinem Falle verzweigt.

– Der Assembler [1] erleichtert erheblich das Schreiben von BCR- und BC-Befehlen: für beide Befehle und für alle sinnvollen Bitkombinationen der Maske verfügt er über sog. "Erweiterte mnemotechnische Operationscodes", die BC- und BCR-Befehle *einschließlich* Maske erzeugen. Zum Beispiel erzeugt der Assembler [1] aus dem mnemotechnischen Operationscode BE einen BC-Befehl mit der Maske B’1000’ (=8), der nach arithmetischen Vergleichen geschrieben werden kann, wenn beim Anzeigewert 0, d.h. bei Gleichheit verzweigt werden soll. Dadurch bleibt dem Autor des Programms das Memorieren obiger Tabelle erspart und wird dem Leser des Programms der Steuerfluß verständlich. Siehe dazu die vollständige Tabelle unter "Erweiterte mnemotechnische Operationscode" im Anhang.

– Man beachte den Unterschied zwischen BCR und BC:
bei BCR wird die Sprungadresse durch den *Inhalt*, bei BC jedoch durch die *Adresse* des zweiten Operanden bestimmt.

Beispiel

Name	Operation	Operanden	
	.		
Beispiel1	CL	4,5	
	BC	7,AGAIN	Sprung bei Anzeige 0
*			durch Maske = 7_{10} = $(0111)_2$
	.		
Beispiel2	TM	SEMAPHOR,X’80’	
	BO	ON	Erweiterter mnemotechnischer
*			Operationscode BO;
*			bewirkt Sprung bei Anzeige =3 durch
*			Maske =1_{10} =$(0001)_2$
	.		
Beispiel3	BR	14	Erweiterter mnemotechnischer
*			Operationscode BR;
*			bewirkt unbedingten Sprung
*			durch Maske =15_{10} =$(1111)_2$.
*			Sprungziel: Adresse im Register 14.
	.		

Branch on Count

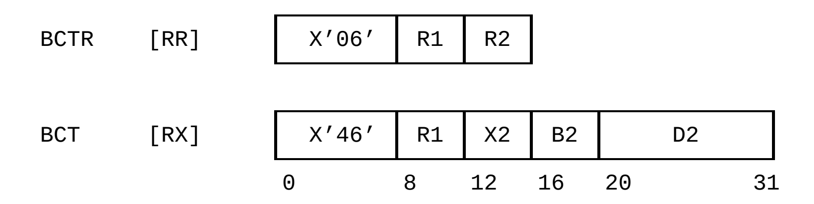
Funktion

Die Befehle BCTR und BCT vermindern den Inhalt eines angegebenen Mehrzweckregisters um Eins und verzweigen an eine angegebene Adresse, wenn der resultierende Registerinhalt ≠0 ist.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	BCTR BCT	R1,R2 R1,D2(X2,B2)	

Maschinenformate



Beschreibung

Von der Binärzahl im Mehrzweckregister R1 wird die Zahl 1 subtrahiert, wobei ein etwaiger Übertrag über die höchstwertige Bitstelle ignoriert wird. Wenn das Resultat ≠0 ist, verzweigt der Befehl BCTR an die im Mehrzweckregister R2 enthaltene Adresse und der Befehl BCR an die Adresse D2(X2,B2). Diese Adresse ist je nach Adressierungsmodus 24 Bit oder 31 Bit lang. Wenn das Resultat =0 ist, wird nicht verzweigt, sondern mit dem nachfolgenden Befehl fortgefahren; ebenso wird nicht verzweigt, wenn beim Befehl BCTR das R2-Feld =0 ist.

Die Sprungadresse wird ermittelt, bevor das Mehrzweckregister R1 verändert wird.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine beim Befehl selbst.
Wenn allerdings tatsächlich verzweigt wird und die Zieladresse keine Halbwortadresse ist oder auf sie nicht zugegriffen werden kann, erfolgt eine entsprechende Programmunterbrechung (mit dem Gewicht $5C_{16}$ bzw. 48_{16}) an der Zieladresse.

Programmierhinweise

– Der Inhalt des Mehrzweckregisters R1 kann gleichermaßen als Zahl mit oder ohne Vorzeichen aufgefaßt werden, weil die Subtraktion von 1 in beiden Fällen das gleiche Resultat liefert.

– Man beachte folgende Grenzfälle: Die Subtraktion von 1 macht aus einem Register R1-Inhalt von -2^{31} den Inhalt $+2^{31}-1$ und aus einem R1-Inhalt von 0 den Inhalt -1: in beiden Fällen wird demzufolge verzweigt. Nicht verzweigt wird nur, wenn das Mehrzweckregister R1 vor dem Befehl BCTR bzw. BCT den Wert +1 enthält (oder wenn beim Befehl BCTR das R2-Feld =0 ist).

– Man beachte folgenden Unterschied zwischen BCTR und BCT: bei BCTR wird die Sprungadresse durch den *Inhalt*, bei BCT jedoch durch die *Adresse* des zweiten Operanden bestimmt.

Beispiel

Name	Operation	Operanden
	.	
	LH	5,=H’100’
AGAIN	EQU	*
	.	
	.	
	.	
	BCT	5,AGAIN
	.	

Im Beispiel wird AGAIN genau 100 mal durchlaufen.

Branch and Set Mode

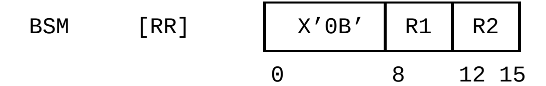
Funktion

Der Befehl BSM speichert den momentanen Adressierungsmodus, setzt danach einen angegebenen Adressierungsmodus und verzweigt in diesem Modus an eine angegebene Adresse.
Die Anzeige wird nicht verändert.
Der Befehl BSM ist nur im Befehlsvorrat der Zentraleinheiten verfügbar, die über den 31-Bit-Adressierungsmodus verfügen.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	BSM	R1,R2	

Maschinenformat



Beschreibung

Wenn das R1-Feld ≠0 ist, wird der momentane Adressierungsmodus in die Bitstelle 0 des Mehrzweckregisters R1 gespeichert; die Bitstellen 1 bis 31 bleiben unverändert. Wenn das R1-Feld =0 ist, wird der momentane Adressierungsmodus nicht gespeichert.

Anschließend wird, wenn das R2-Feld ≠0 ist, der Adressierungsmodus gesetzt, der sich aus dem Bit 0 des Mehrzweckregisters R2 ergibt und es wird in diesem Modus an die in den Bitstellen 1 bis 31 von R2 enthaltene Adresse verzweigt. Diese Adresse ist je nach Adressierungsmodus 24 Bit oder 31 Bit lang. Wenn das R2-Feld =0 ist, geschieht dies nicht, sondern es wird im momentanen Adressierungsmodus nach dem Befehl BSM fortgefahren.

In beiden Registern R1 und R2 bedeutet ein Wert 0 an der Bitstelle 0 den 24-Bit-Adressierungsmodus und ein Wert 1 den 31-Bit-Adressierungsmodus.

Die Sprungadresse wird ermittelt, bevor das Mehrzweckregister R1 verändert wird.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine beim Befehl selbst.
Wenn allerdings die Zieladresse keine Halbwortadresse ist oder auf sie nicht zugegriffen werden kann, erfolgt eine entsprechende Programmunterbrechung (mit dem Gewicht $5C_{16}$ bzw. 48_{16}) an der Zieladresse.

Programmierhinweise

– Der Befehl BSM dient hauptsächlich zum Rücksprung aus einem Unterprogramm, das mit einem der Befehle BASR, BAS oder BASSM aufgerufen wurde. Der Befehl kann jedoch auch als unbedingter Sprung in anderen Fällen verwendet werden.

– Es muß davor gewarnt werden, den Befehl BSM zum Rücksprung aus einem Unterprogramm zu verwenden, das mit dem Befehl BAL aufgerufen wurde. Wenn nämlich das aufrufende Programm im 24-Bit-Adressierungsmodus abläuft, kann im aufrufenden Programm nach der Rückkehr falscher Adressierungsmodus herrschen (wegen des ILC-Wertes $(10)_2$ in den Bitstellen 0 und 1 der Befehlsfolgeadresse). Unterprogramme, die mit BALR oder BAL gerufen wurden, sollten mit den Befehl BCR oder BC verlassen werden.

– Der Befehl BSM ist nur auf den Zentraleinheiten verfügbar, die über den 31-Bit-Adressierungsmodus verfügen. Um ein Assemblerprogramm unabhängig von der Zentraleinheit zu machen, auf der es abläuft, gibt es im Vorrat der Makroaufrufe des BS2000 (V9.0 und Nachfolger) den Makro ##BSM, mit dem der Befehl BSM "eingeschalt" wird. Der Makroaufruf ##BSM generiert auf Anlagen mit 31-Bit-Adressierungsmodus den Befehl BSM und auf Anlagen, die nur über den 24-Bit-Adressierungsmodus verfügen, den Befehl BR.
  Eine vollständige Darstellung der Adressierungsmodi und der verschiedenen Programmverknüpfungsformen enthält das Handbuch "Einführung in die XS-Programmierung (für ASSEMBLER-Programmierer)" [2].

Beispiel

Siehe für die Verwendung von BSM das Beispiel unter BASSM.

Branch on Index

Funktion

Die Befehle BXH und BXLE addieren den Inhalt eines Mehrzweckregisters zum Inhalt eines anderen Mehrzweckregisters vorzeichengerecht und verzweigen an eine angegebene Adresse, wenn die resultierende Summe größer (BXH) bzw. gleich oder kleiner (BXLE) ist als der Inhalt eines dritten Mehrzweckregisters.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	BXH BXLE	R1,R3,D2(B2) R1,R3,D2(B2)	

Maschinenformate

		X'..'	R1	R3	B2	D2
BXH	[RS]	X'86'	R1	R3	B2	D2
BXLE	[RS]	X'87'	R1	R3	B2	D2
		0	8	12	16	20 31

Beschreibung

Zunächst werden die 32 Bit lange Festpunktzahl im Mehrzweckregister R1 und die 32 Bit lange Festpunktzahl im Mehrzweckregister R3 addiert. Die Summe wird vorzeichengerecht als 32 Bit lange Festpunktzahl gebildet, aber ein etwaiger Übertrag über die höchstwertige Binärstelle wird ignoriert.

Die Summe wird mit der 32 Bit langen Festpunktzahl eines Vergleichsregisters vorzeichengerecht verglichen. Nach dem Vergleich wird die Summe in das Mehrzweckregister R1 gespeichert. Das Vergleichsregister ist das Mehrzweckregister R3+1, wenn R3 geradzahlig ist, andernfalls das Mehrzweckregister R3.

Anschließend wird an die Adresse D2(B2) verzweigt, wenn die Summe größer (bei BXH) bzw. gleich oder kleiner (bei BXLE) ist als der Inhalt des Vergleichsregisters; andernfalls wird nicht verzweigt und mit dem nächsten Befehl fortgefahren.

Der Vergleich wird durchgeführt und die Sprungadresse D2(B2) wird ermittelt, bevor das Mehrzweckregister R1 verändert wird.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine beim Befehl selbst.
Wenn allerdings tatsächlich verzweigt wird und die Zieladresse keine Halbwortadresse ist oder auf sie nicht zugegriffen werden kann, erfolgt eine entsprechende Programmunterbrechung (mit dem Gewicht $5C_{16}$ bzw. 48_{16}) an der Zieladresse.

Programmierhinweise

- Wenn R3 nicht geradzahlig ist, dient sein Inhalt sowohl als Inkrementwert für das Mehrzweckregister R1 als auch als Vergleichswert.
- Für R1 und R3 darf das Mehrzweckregister 0 verwendet werden.
- Wenn R1=R3 ist, wird das Inkrement verdoppelt. Wenn R1 außerdem ungeradzahlig ist, ist zu berücksichtigen, daß der Vergleich *vor* der Veränderung des Mehrzweckregisters R1 erfolgt, daß also der doppelte R1-Inhalt mit dem einfachen R1-Inhalt verglichen wird und erst anschließend der doppelte R1-Inhalt den einfachen R1-Inhalt ersetzt.
- Die Befehle BXH und BXLE leisten gute Dienste bei der Programmierung von "for"-Schleifen, bei denen eine Laufvariable von einem Anfangswert über ein konstantes Inkrement (bzw. Dekrement) bis zu einem Endwert wächst (bzw. abnimmt). In diesen Fällen hält man das Inkrement (bzw. Dekrement) in einem geradzahligen Register R3 und den Endwert im benachbarten ungeradzahligen Register R3+1 und verwendet für den Anfangswert und die Laufvariable ein (beliebiges) anderes Register R1.
- Wenn das Schleifenproblem so arrangiert werden kann, daß die Laufvariable von einem Anfangswert >0 über ein konstantes Dekrement auf den Endwert =0 zuläuft, läßt sich sogar noch ein Mehrzweckregister sparen: man verwendet BXH und trägt das Dekrement (als negatives Inkrement) in ein *ungeradzahliges* Mehrzweckregister ein; der Befehl BXH verwendet dann zur Dekrementierung und zum Vergleich dasselbe Register.

Beispiele

Beispiel 1

Der in den "Programmierhinweisen" zuerst genannte Fall einer Schleife mit einer Laufvariablen, die vom Anfangswert a über die Werte a+i, a+2i, ... , zum Endwert z wächst, läßt sich mittels BXLE z.B. so programmieren:

Name	Operation	Operanden
	.	
	LA	3,*a* beliebiges Register für die Laufvariable
	LA	8,*i* geradzahliges Register für das Inkrement
	LA	9,*z* Nachbar-Register für den Endwert
BODY	EQU	* } Schleifenrumpf
	.	
	.	
	.	
	BXLE	3,8,BODY
	.	

Dabei ist zu beachten, daß BODY auch noch für den Laufvariablen-Wert z selbst ausgeführt wird.

Beispiel 2

Der in den "Programmierhinweisen" zweitgenannte Fall einer Schleife mit einer Laufvariablen, die von einem Anfangswert a zum Endwert 0 über die Werte a-i, a-2i, ... abnimmt, kann unter Verwendung des Befehls BXH so aussehen:

Name	Operation	Operanden
	.	
	LA	3,*a* beliebiges Register für die Laufvariable
	LH	9,=H'-*i*' ungeradzahliges Register für
*		Dekrement <u>und</u> Vergleichswert
BODY	EQU	* } Schleifenrumpf
	.	
	.	
	.	
	BXH	3,9,BODY
	.	

Man vergleiche das Operandenfeld von BXH in diesem Beispiel mit dem von BXLE im vorigen Beispiel.
Bei der letzten Ausführung des BXH wird -i mit -i verglichen und demzufolge nicht mehr verzweigt, während bei den nicht-letzten Ausführungen des BXH der Wert 0 mit -i verglichen wird und folglich ein Rücksprung erfolgt. Der Vergleich mit dem Endwert erfolgt bei BXH (und BXLE) vorzeichengerecht, so daß der Vergleich einer positiven Laufvariablen mit einem negativen Endwert das gewünschte Ergebnis liefert. (Allzusehr kann man diesen "Trick" wohl nicht empfehlen, aber er ist legal.)

Compare

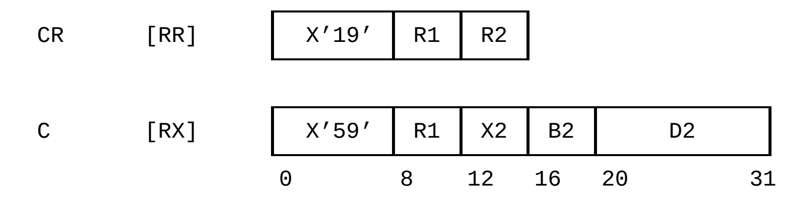
Funktion

Die Befehle CR und C vergleichen zwei 32 Bit lange Festpunktzahlen vorzeichengerecht. Die Anzeige wird gemäß dem Vergleichsergebnis gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	CR C	R1,R2 R1,D2(X2,B2)	 D2(X2,B2): Wortgrenze

Maschinenformate



Beschreibung

Durch den Befehl CR wird der Inhalt des Mehrzweckregisters R2, durch den Befehl C wird das mit D2(X2,B2) adressierte Wort des Hauptspeichers vorzeichengerecht mit dem Inhalt des Mehrzweckregisters R1 verglichen. Beide Operanden werden als 32 Bit lange Binärzahlen mit Vorzeichen (Festpunktzahlen) behandelt.

Der Inhalt des Mehrzweckregisters R1 wird nicht verändert.

Anzeige

0~Equal	Operand1 = Operand2
1~Low	Operand1 < Operand2
2~High	Operand1 > Operand2
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Adreßfehler	X'48' X'5C'	C: Lesezugriff auf Operand2 unmöglich C: D2(X2,B2) keine Wortgrenze.

Beispiel

Name	Operation	Operanden
	. L C .	 5,=F'-1' 5,=F'+1' setzt Anzeige 1~Low

Anstelle des Befehls C würde der Befehl CL die Anzeige 2~High setzen.

Compare Halfword

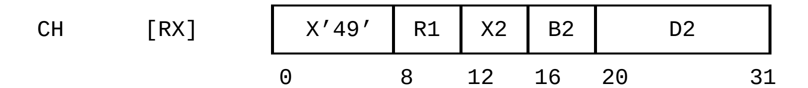
Funktion

Der Befehl CH vergleicht eine 32 Bit lange Festpunktzahl vorzeichengerecht mit einer 16 Bit langen Festpunktzahl.
Die Anzeige wird gemäß dem Vergleichsergebnis gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	CH	R1,D2(X2,B2)	D2(X2,B2): Halbwortgrenze

Maschinenformat



Beschreibung

Der Inhalt des Mehrzweckregisters R1 wird mit dem durch D2(X2,B2) adressierten Halbwort des Hauptspeichers vorzeichengerecht verglichen. Der Register-Operand wird als 32 Bit lange Festpunktzahl, der Halbwort-Operand wird als 16 Bit lange Festpunktzahl behandelt.

Anzeige

0~Equal Operand1 = Operand2
1~Low Operand1 < Operand2
2~High Operand1 > Operand2
3 Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	D2(X2,B2) keine Halbwortgrenze.

Beispiel

Name	Operation	Operanden
	.	
	L	5,=F’-1’
	CH	5,=H’-1’ setzt Anzeige 0~Equal
	.	

Im Beispiel wird eine Binärzahl aus 32 Einsen mit einer Binärzahl aus 16 Einsen verglichen. Der Vergleich ergibt "gleich", weil der zweite Operand so behandelt wird, als sei er links mit 16 Einsen aufgefüllt.

Compare Logical

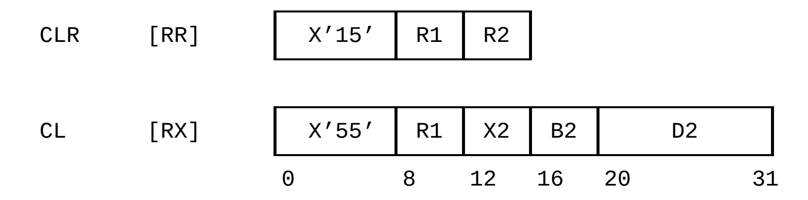
Funktion

Die Befehle CLR und CL vergleichen zwei 32 Bit lange Binärzahlen logisch.
Die Anzeige wird gemäß dem Vergleichsergebnis gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	CLR CL	R1,R2 R1,D2(X2,B2)	 D2(X2,B2): Wortgrenze

Maschinenformate



Beschreibung

Durch den Befehl CLR wird der Inhalt des Mehrzweckregisters R2, durch den Befehl CL wird das mit D2(X2,B2) adressierte Wort des Hauptspeichers logisch mit dem Inhalt des Mehrzweckregisters R1 verglichen. Beide Operanden werden als 32 Bit lange Binärzahlen ohne Vorzeichen behandelt.

Der Inhalt des Mehrzweckregisters R1 wird nicht verändert.

Anzeige

0~Equal	Operand1 = Operand2
1~Low	Operand1 < Operand2
2~High	Operand1 > Operand2
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Adreßfehler	X’48’ X’5C’	CL: Lesezugriff auf Operand2 unmöglich CL: D2(X2,B2) keine Halbwortgrenze.

Beispiel

Name	Operation	Operanden
	. L CL .	 5,=F’-1’ 5,=F’+1’ setzt Anzeige 2~High

Anstelle des Befehls CL würde der Befehl C die Anzeige 1~Low setzen.

Compare Logical Characters

Funktion

Der Befehl CLC vergleicht zwei Zeichenfelder logisch.
Die Anzeige wird gemäß dem Vergleichsergebnis gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	CLC	D1(L,B1),D2(B2)	1 ≤ L ≤ 256

Maschinenformat

CLC [SS]

X'D5'	L-1	B1	D1	B2	D2
0	8	16	20	32	36 47

Beschreibung

Das durch D1(B1) adressierte Zeichenfeld im Hauptspeicher der Länge L Byte wird logisch von links nach rechts mit dem durch D2(B2) adressierten Zeichenfeld gleicher Länge verglichen. Der Befehl wird bei Ungleichheit beendet oder wenn die Operanden abgearbeitet sind.

Anzeige

0~Equal Operand1 = Operand2
1~Low Operand1 < Operand2
2~High Operand1 > Operand2
3 Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	Lesezugriff auf Operand1 oder Operand2 unmöglich

Programmierhinweise

- Mit dem Befehl CLC ist ein Vergleich bis zu einer Feldlänge von 256 Byte möglich. Für größere Feldlängen steht der Befehl CLCL zur Verfügung.
- Die zu vergleichenden Zeichenfelder dürfen sich in beliebiger Weise überlappen. Man kann sich dies zunutze machen, um z.B. ein Zeichenfeld daraufhin zu überprüfen, ob in ihm ein Sub-Zeichenfeld wiederholt vorkommt (siehe Beispiel3).

Beispiele

Name	Operation	Operanden
	.	
FIELD1	DC	C’AC’
FIELD2	DC	C’AB’
FIELD3	DC	C’*********’
	.	
	.	
	.	
Beispiel1	CLC	FIELD1(1),FIELD2 setzt die Anzeige 0~Equal
	.	
	.	
Beispiel2	CLC	FIELD1,FIELD2 setzt die Anzeige 2~High
	.	
	.	
Beispiel3	CLC	FIELD3+1(L’FIELD3-1),FIELD3
*		
*		FIELD3 wird getestet, ob es
*		lauter gleiche Zeichen enthält
*		setzt die Anzeige 0~Equal
	.	

Compare Logical Long

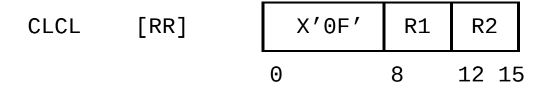
Funktion

Der Befehl CLCL vergleicht den Inhalt eines Hauptspeicherbereichs logisch von links nach rechts mit einem anderen Hauptspeicherbereich. Die beiden Bereiche können bis zu 2^{24} Byte, d.h. bis zu 16 MB lang sein.
Die Anzeige wird gemäß dem Vergleichsergebnis gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	CLCL	R1,R2	R1 und R2 geradzahlig

Maschinenformat

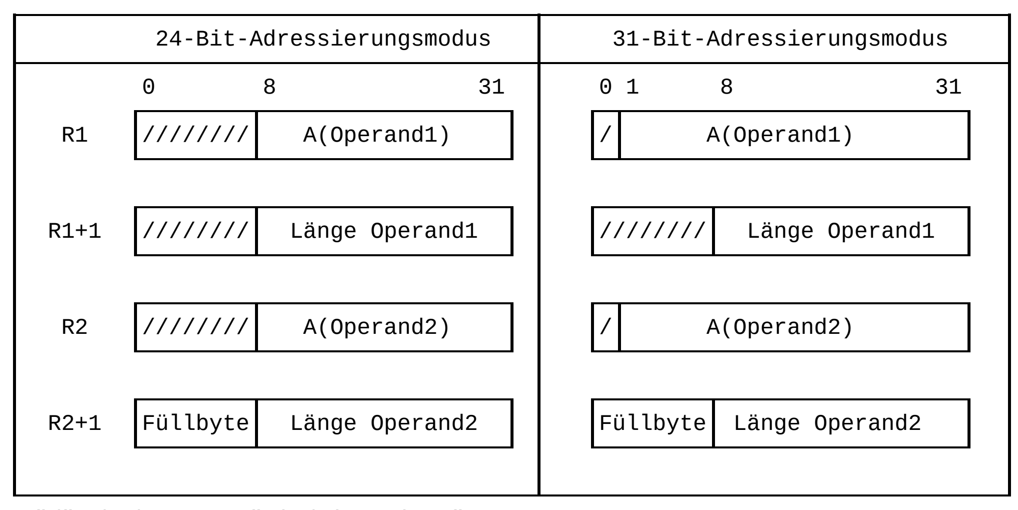


Beschreibung

R1 und R2 bestimmen jeweils ein Paar von Registern, bestehend aus den Mehrzweckregistern R1 und R1+1 bzw. R2 und R2+1. R1 und R2 müssen beide geradzahlig sein, andernfalls erfolgt eine Programmunterbrechung wegen Adreßfehlers.

Der Operand1 bzw. der Operand2 sind durch die Registerpaare R1 und R1+1 bzw. R2 und R2+1 bestimmt. Ihre Anfangsadressen werden dem ersten Register R1 bzw. R2, ihre Längen (in Byte) werden dem zweiten Register R1+1 bzw. R2+1 entnommen. Das Register R2+1 enthält außerdem die Codierung des Füllbyte.

Die Adressendarstellung in R1 bzw. R2 ist abhängig vom Adressierungsmodus.
Es gilt folgende Zuordnung:



"/" bedeutet: "wird ignoriert"

Der Vergleich erfolgt logisch von links nach rechts. Er endet, wenn entweder Ungleichheit festgestellt wird oder das Ende des längeren Operanden erreicht ist. Wenn die Operanden verschiedene Länge haben, wird der kürzere Operand so behandelt, als sei er rechts mit Füllbytes auf die Länge des längeren Operanden aufgefüllt. Die Codierung dieser Füllbytes wird dabei dem höchstwertigen Byte von R2+1 entnommen.

Der Befehl CLCL ist hardwareseitig unterbrechbar. Bei einer Unterbrechung wird der bis dahin erreichte Vergleichsfortschritt in den Registerpaaren R1 und R2 festgehalten (durch Abspeicherung der hochgezählten Adressen und der heruntergezählten Längen). Nach der Unterbrechung wird dann der Vergleich an der unterbrochenen Hauptspeicherstelle fortgesetzt.

Nach der Befehlsausführung enthalten die Registerpaare R1 und R1+1 bzw. R2 und R2+1 folgende Werte:

- Wenn die beiden Operanden gleich sind, enthalten die Register R1 und R2 die um die Längenfelder in R1+1 bzw. R2+1 erhöhten Adressen des ersten Operanden bzw. des zweiten Operanden; die Längenfelder in R1+1 bzw. R2+1 enthalten den Wert 0.
- Wenn die Operanden ungleich sind und die Ungleichheit nicht im Bereich der Füllbytes aufgetreten war, enthält R1 und R2 jeweils die Adresse des ersten ungleichen Byte im ersten Operanden bzw. zweiten Operanden; die Längenfelder in R1+1 bzw. R2+1 sind um die Anzahl der gleichen Byte vermindert.

– Wenn die Operanden ungleich sind, aber die Ungleichheit erst im Bereich der Füllbyte aufgetreten war, dann ist das erste Register des längeren Operanden um die Anzahl der ’gleichen’ Byte und das erste Register des kürzeren Operanden um dessen Länge erhöht; im jeweils zweiten Register ist das Längenfeld des längeren Operanden dementsprechend um die Anzahl der ’gleichen’ Byte vermindert und das des kürzeren Operanden ist =0 gesetzt.

In allen drei Fällen sind nach der Befehlsausführung jeweils die höchstwertigen Byte von R1+1 und R2+1 unverändert und die oberen acht Bit bzw. das obere Bit (je nach Adressierungsmodus) von R1 und R2 =0 gesetzt.

Anzeige

0~Equal	Operand1 = Operand2 oder L’Operand1=0 und L’Operand2=0
1~Low	Operand1 < Operand2
2~High	Operand1 > Operand2
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lesezugriff auf Operand1 oder Operand2 unmöglich.
Adreßfehler	X’5C’	R1 oder R2 nicht geradzahlig.

Programmierhinweise

– Wenn beide Operanden die Länge 0 haben, werden sie als gleich betrachtet.

– Die beiden Operanden dürfen sich in beliebiger Weise überlappen.

– Das Anwenderprogramm sollte von sich aus sicherstellen, daß die gesamten Adreßbereiche beider Operanden ausschließlich in seinem eigenen Adreßraum liegen. Der Befehl CLCL prüft nämlich erst während seines Ablaufs und nicht bereits an seinem Beginn, ob die Adreßbereiche in erlaubten Grenzen liegen. Wenn ein unerlaubter Teil-Bereich entdeckt wird, bricht der Befehl nicht notwendig ab, sondern setzt ggf. hinter dem Teil-Bereich fort, aber in jedem Fall sind die Ergebnisse sowohl im Hauptspeicher wie in der Anzeige unbrauchbar.

– Der Anwender sollte den Befehl CLCL nicht mit dem Befehl EX auslösen, wenn dieser EX die gleichen Register benutzt wie der CLCL.

Beispiel

Die folgenden Befehle vergleichen maximal 20000 Byte der Hauptspeicherbereiche C1 und C2. Da C2 nur eine Länge von 15000 Byte hat, werden die letzten 5000 Byte von C1 gegen das Füllbyte verglichen, sofern die ersten 15000 Byte gleich sind.

Name	Operation	Operanden
	.	
	LM	4,5,=A(C1,20000) Register 4 und 5: Operand1
	LM	10,11,=A(C2,15000) Register 10 und 11: Operand2
	ICM	11,B'1000',=C' ' Füllbyte nach Byte 0 von Reg 11
	CLCL	4,10
	.	

Nach der Befehlsausführung von CLCL sind folgende Werte möglich:

- im Falle von Gleichheit enthalten die Register 4 und 10 die Werte A(C1+20000) und A(C2+15000) und die Längenfelder der Register 5 und 11 den Wert 0
- im Falle von Ungleichheit innerhalb der ersten 15000 Byte zeigen die Register 4 und 10 auf das erste ungleiche Byte in C1 bzw. C2 und sind die Längenfelder der Register 5 und 11 um die Anzahl der gleichen Byte vermindert
- im Falle von Ungleichheit im Bereich der letzten 5000 Byte enthält das Register 4 die Adresse des ersten ungleichen Byte und ist das Längenfeld des Registers 5 um die Anzahl der gleichen Byte vermindert, aber das Register 10 enthält den Wert A(C2+15000) und das Längenfeld von Register 11 den Wert 0.

In allen Fällen ist das obere Byte von Register 5 und 11 unverändert, d.h. im Beispielfalle =X'00' bzw. =C'␣'.

Compare Logical Immediate

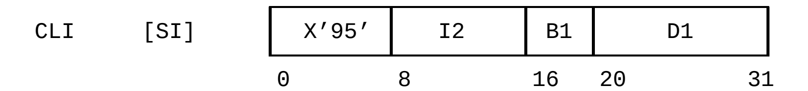
Funktion

Der Befehl CLI vergleicht ein Byte des Hauptspeichers mit dem Direktoperanden I2.
Die Anzeige wird gemäß dem Vergleichsergebnis gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	CLI	D1(B1),I2	X’00’ ≤ I2 ≤ X’FF’

Maschinenformat



Beschreibung

Das mit D1(B1) adressierte Byte des Hauptspeichers wird logisch mit dem Direktoperanden I2 verglichen.

Anzeige

0~Equal Operand1 = I2
1~Low Operand1 < I2
2~High Operand1 > I2
3 Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lesezugriff auf Operand1 unmöglich

Beispiel

```
Name        | Operation | Operanden
------------+-----------+------------------------------------------------------
            | .         |
FIELD1      | DC        | C'AC'
            | .         |
            | .         |
            | .         |
            | CLI       | FIELD1,C'A'           setzt die Anzeige 0~Equal
            | .         |
```

Compare Logical under Mask

Funktion

Der Befehl CLM vergleicht ausgewählte Byte eines Mehrzweckregisters logisch mit einem Zeichenfeld im Hauptspeicher.
Die Anzeige wird gemäß dem Vergleichsergebnis gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	CLM	R1,M3,D2(B2)	B’0000’ ≤ M3 ≤ B’1111’

Maschinenformat

CLM [RS]

X’BD’	R1	M3	B2	D2
0	8	12	16	20 31

Beschreibung

Die 4 Bit der "Maske" M3 entsprechen eins zu eins den 4 Byte des Mehrzweckregisters R1 (von links nach rechts sowohl in der Maske wie im Register). Diejenigen Byte in R1, denen Einsen in der Maske entsprechen, werden als zusammenhängendes Feld betrachtet; dieses Feld wird mit dem durch D2(B2) adressierten Zeichenfeld des Hauptspeichers logisch verglichen.

Anzeige

0~Zero Die ausgewählten Byte von R1 sind = Zeichenfeld oder die Maske ist $=0_{16}$.
1~Minus Die ausgewählten Byte von R1 sind < Zeichenfeld.
2~Plus Die ausgewählten Byte von R1 sind > Zeichenfeld.
3 Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lesezugriff auf Operand2 unmöglich, auch wenn M3 = 0_{16} ist.

Programmierhinweise

- Die Länge in Byte des Hauptspeicherfelds ist gleich der Anzahl der Einsen in der Maske.
- Bei Verwendung einer Maske aus lauter Einsen (B’1111’) leistet der Befehl CLM das gleiche wie der Befehl CL, aber das Hauptspeicherfeld braucht nicht an einer Wortgrenze ausgerichtet zu sein.

Beispiel

Name	Operation	Operanden
	. CLM .	3,B’1001’,=C’LR’

Der Beispielbefehl vergleicht logisch das höchstwertige und das niedrigstwertige Byte des Mehrzweckregisters 3 mit der Zeichenfolge LR.

Compare and Swap

Funktion

Die Befehle CS und CDS speichern den Inhalt eines Mehrzweckregisters in einen Hauptspeicherbereich, sofern der Inhalt dieses Hauptspeicherbereichs gleich ist dem Inhalt eines anderen Mehrzweckregisters. Die Befehle wahren die Integrität ihrer Daten während der Befehlsausführung.
Die Anzeige wird gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	CS	R1,R3,D2(B2)	D2(B2): Wortgrenze
	CDS	R1,R3,D2(B2)	R1, R3 geradzahlig und
*			D2(B2): Doppelwortgrenze

Maschinenformate

CS	[RS]	X'BA'	R1	R3	B2	D2	(Kurze Operanden)
CDS	[RS]	X'BB'	R1	R3	B2	D2	(Lange Operanden)
		0	8	12	16	20 31	

Beschreibung

Der erste Operand (R1) wird mit dem zweiten Operanden (D2(B2)) logisch verglichen. Bei Gleichheit ersetzt der dritte Operand (R3) den zweiten Operanden und die Anzeige wird auf 0~Equal gesetzt, bei Ungleichheit ersetzt der zweite Operand den ersten Operanden und die Anzeige wird auf 1~Not Equal gesetzt. Der zweite Operand bleibt bis zum Abschluß des Befehls gegen jeden anderen Schreibzugriff gesperrt.

Bei CS sind alle drei Operanden 32 Bit lang, bei CDS sind sie 64 Bit lang.

Befehl	Operand1	Operand2	Operand3
CS	Register R1	Wort bei D2(B2)	Register R3
CDS	Registerpaar R1, R1+1	Doppelwort bei D2(B2)	Registerpaar
*			R3, R3+1

Anzeige

0~Equal	Operand2 war gleich Operand1; Operand2 ist durch Operand3 ersetzt.
1~Not Equal	Operand2 war ungleich Operand1; Operand1 ist durch Operand2 ersetzt.
2	Nicht verwendet.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lese-Schreibzugriff auf Operand2 unmöglich.
Adreßfehler	X’5C’	CS: D2(B2) keine Wortgrenze. CDS: D2(B2) keine Doppelwortgrenze oder R1 oder R3 nicht geradzahlig

Programmierhinweise

– Die Befehle CS und CDS (sowie TS) sperren als einzige Befehle des Befehlsvorrats während der Zeit ihrer Ausführung die von ihnen adressierten Hauptspeicherdaten gegen Lese- und Schreibzugriffe durch andere Programme derselben oder anderer Zentraleinheiten. Sie verhindern auch, daß während ihres Ablaufs auf nachfolgende Befehle und/oder deren Operanden zugegriffen wird. Die Befehle stellen also sicher, daß der Datenzustand zum Zeitpunkt ihres (anfänglichen) Lesevorgangs gleich ist dem Zustand zum Zeitpunkt ihres (abschließenden) Schreibvorgangs. Zu diesem Zweck findet vor und nach dem Befehl CS und CDS in der Hardware eine sog. Serialisierung statt, bei der alle anstehenden Speicherzugriffe abgearbeitet werden. Dieser Mechanismus prädestiniert die Befehle CS und CDS für Synchronisationsprobleme in Multiprozessor-Anwendungen.

  In solchen Anwendungsfällen muß jedes der simultan ablaufenden Programme berücksichtigen, daß während seiner eigenen Bearbeitung und Veränderung eines gemeinsamen Speicherbereichs ein anderes Programm möglicherweise das Gleiche tut und dadurch sich die Bearbeitungs-Resultate gegenseitig auslöschen können.

  Man betrachte dazu den einfachen Fall, bei dem zwei zeitgleich ablaufende Programme A und B ein gemeinsames Hauptspeicherwort *Zähler* um Eins erhöhen.

Wenn beide Programme diese Erhöhung zufälligerweise gleichzeitig tun, aber Programm B um einen Befehl gegenüber Programm A versetzt abläuft, dann kann folgender Effekt auftreten:

Zeit	Zähler	Programm A	Programm B
t_0	sei 100	Liest Zähler (100)	
t_1	100	erhöht Zähler (101)	Liest Zähler (100)
t_2	101	speichert Zähler (101)	erhöht Zähler (101)
t_3	101		speichert Zähler (101)

Obwohl beide Programme erhöht haben, enthält *Zähler* letztlich statt des Werts 102 nur den Wert 101, weil Programm A noch nicht gespeichert hatte als Programm B schon begann.

Weiter unten wird eine Befehlsfolge gezeigt, die eine sichere Methode gegen diesen Effekt darstellt. Die Idee ist, zur Speicherung des geänderten Werts den Befehl CS (oder CDS) statt z.B. den Befehl ST zu verwenden. Der CS-Befehl stellt nämlich vor dem Speichern zum Zeitpunkt t_3 des Programms B fest, daß der momentane Inhalt von *Zähler* nicht mehr gleich ist seinem Inhalt zum Zeitpunkt t_1, weil das Programm A ihn inzwischen, nämlich zum Zeitpunkt t_2 geändert hat. Programm B speichert dann eben nicht, sondern wiederholt die Erhöhung, wobei es korrekterweise vom Wert 101 ausgeht.

– Auch wenn ein gemeinsam benutzter Hauptspeicherbereich länger als 4 bzw. 8 Byte ist, können die Befehle CS und CDS verwendet werden. Dazu legt man gewöhnlich ein den Speicherbereich stellvertretendes Wort oder Doppelwort an, in welchem man die möglichen "Zustände" des Speicherbereichs festhält. Mittels CS oder CDS wird dann statt des eigentlichen Hauptspeicherbereichs nur das Zustandswort verwaltet.

– Die Befehle CS und CDS sollten *nur* zur Koordinierung von Programmen (im selben oder in verschiedenen Zentraleinheiten) verwendet werden und nicht zum Ersatz einer Befehlsfolge aus CL- und ST-Befehlen: CS und CDS sind nämlich zeitaufwendig und blockieren die Befehlsausführung in anderen Zentraleinheiten.

Beispiel

Eine sichere Methode zum Update eines gemeinsam genutzten Hauptspeicherworts (SHAREWD) mittels CS besteht in folgender Prinzip-Befehlsfolge:

Name	Operation	Operanden
	.	
UPDATE	L	R1,SHAREWD
AGAIN	LR	R3,R1
	.	

< ermitteln Updatewert in R3, R1 muß unverändert bleiben >

	.	
	CS	R1,R3,SHAREWD
	BNE	AGAIN
	.	

Die Befehlsfolge beginnt mit dem Laden des Anfangswerts von SHAREWD in das Mehrzweckregister R1; dort muß es bis zum Befehl CS unverändert bleiben. Der Update-Wert von SHAREWD wird in einem anderen Register (R3) hergestellt. Die abschließende Speicherung erfolgt durch den Befehl CS, der zunächst prüft, ob der momentane Wert von SHAREWD (noch) gleich ist seinem Anfangswert (in R1). Nur wenn dies der Fall ist, speichert CS tatsächlich; er setzt außerdem die Anzeige 0~Equal, wodurch die Befehlsfolge verlassen wird. Wenn aber zum Ausführungszeitpunkt von CS der Wert von SHAREWD nicht (mehr) mit dem Inhalt von R1 übereinstimmt, erfolgt keine Speicherung, stattdessen wird der Inhalt des inzwischen geänderten Hauptspeicherworts ins Register R1 geladen, die Anzeige 1~Not Equal gesetzt und das Programmstück wiederholt.

Convert to Binary

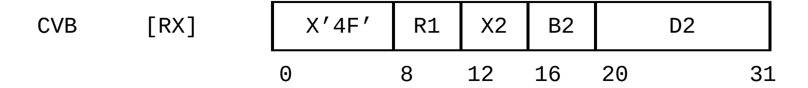
Funktion

Der Befehl CVB konvertiert eine gepackte Dezimalzahl in eine 32 Bit lange Festpunktzahl.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	CVB	R1,D2(X2,B2)	D2(X2,B2): Doppelwortgrenze

Maschinenformat



Beschreibung

D2(X2,B2) muß eine genau 8 Byte lange gepackte Dezimalzahl in einem Doppelwort des Hauptspeichers adressieren. Diese wird in eine 32 Bit lange Festpunktzahl mit Vorzeichen konvertiert und in das Mehrzweckregister R1 gespeichert.

Die zu konvertierende Dezimalzahl muß im Bereich -2^{31} ... $+2^{31}-1$ liegen, d.h. muß mindestens -2147483648 und darf höchstens +2147483647 betragen. Wenn diese Bedingung nicht erfüllt ist, wird die Konvertierung trotzdem ausgeführt, aber das Mehrzweckregister R1 enthält nach der Befehlsausführung nur die niedrigstwertigen 32 Bit der Festpunktzahl und es erfolgt außerdem eine Programmunterbrechung wegen Divisionsfehlers.

Die zu konvertierende Dezimalzahl wird auf korrektes, gepacktes Format geprüft. Im Fehlerfalle erfolgt eine Programmunterbrechung wegen Datenfehlers.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	Lesezugriff auf Operand2 unmöglich
Adreßfehler	X'5C'	D2(X2,B2) keine Doppelwortgrenze
Divisionsfehler	X'68'	Die zu konvertierende Dezimalzahl ist >+2147483647 oder <-2147483648
Datenfehler	X'60'	Die zu konvertierende Zahl ist keine korrekt gepackte, 8 Byte lange Dezimalzahl

Programmierhinweise

– Wenn die Dezimalzahl negativ ist, ist die Festpunktzahl durch ihr Zweierkomplement dargestellt.

Beispiele

Die folgenden Beispiele liefern folgende Ergebnisse im Mehrzweckregister 3:

FIELD (an Doppelwortgrenze)	Beispielbefehl	Register 3 nachher
PL8'255'	CVB 3,FIELD	F'255' = X'000000FF'
PL8'-255'	CVB 3,FIELD	F'-255' = X'FFFFFF01'
PL8'+2147483647'	CVB 3,FIELD	F'2147483647' = X'7FFFFFFF'
PL8'-2147483649'	CVB 3,FIELD	F'2147483647' = X'7FFFFFFF'

Bei allen Beispielen ist unterstellt, daß der Hauptspeicher-Operand FIELD an einer Doppelwortgrenze ausgerichtet ist.

Beim letzten Beispiel tritt eine Programmunterbrechung wegen Divisionsfehlers auf, weil die zu konvertierende Dezimalzahl (um Eins) zu klein ist. Im Register 3 stehen die niedrigstwertigen 32 Bit der korrekten Festpunktzahl, die hier gleich sind der Festpunktzahl aus der größtmöglichen Dezimalzahl.

Convert to Decimal

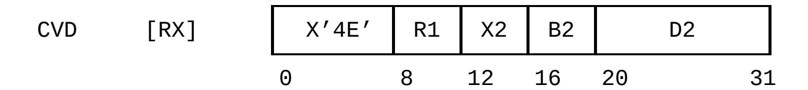
Funktion

Der Befehl CVD konvertiert eine 32 Bit lange Festpunktzahl in eine gepackte, 8 Byte lange Dezimalzahl.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	CVD	R1,D2(X2,B2)	D2(X2,B2): Doppelwortgrenze

Maschinenformat



Beschreibung

Die Festpunktzahl (mit Vorzeichen) im Mehrzweckregister R1 wird in eine genau 8 Byte lange, 15-stellige gepackte Dezimalzahl konvertiert und in das mit D2(X2,B2) adressierte Doppelwort des Hauptspeichers übertragen. Das Mehrzweckregister R1 bleibt unverändert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	Schreibzugriff auf Operand2 unmöglich
Adreßfehler	X'5C'	D2(X2,B2) keine Doppelwortgrenze

Programmierhinweise

- Es kann jede 32 Bit lange, vorzeichenbehaftete Festpunktzahl konvertiert werden.
- Wenn die Dezimalzahl positiv ist, wird ihr Vorzeichen = C_{16} gesetzt, sonst = D_{16}.

Beispiele

Die folgenden Beispiele liefern folgende Ergebnisse in FIELD (das an einer Doppelwort-grenze ausgerichtet sein muß):

Register 3	Beispielbefehl	FIELD (an D’wortgrenze)
F’255’	CVD 3,FIELD	PL8’255’
F’-255’	CVD 3,FIELD	PL8’-255’
X’FFFFFFFF’	CVD 3,FIELD	PL8’-1’
X’80000000’	CVD 3,FIELD	PL8’-2147483648’

Divide

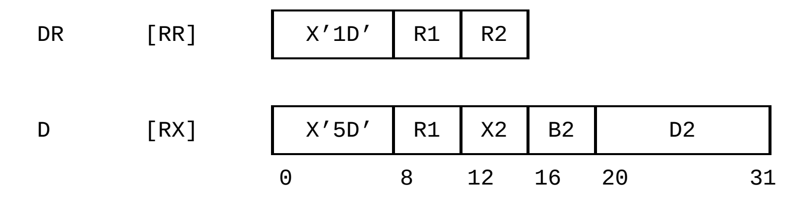
Funktion

Die Befehle DR und D dividieren eine 64 Bit lange Festpunktzahl durch eine 32 Bit lange Festpunktzahl vorzeichengerecht. Der Rest und der Quotient ersetzen den Dividenden.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	DR	R1,R2	R1 geradzahlig
	D	R1,D2(X2,B2)	R1 geradzahlig und
*			D2(X2,B2): Wortgrenze

Maschinenformate



Beschreibung

Das R1-Feld der Befehle DR und D bestimmt ein Paar von Mehrzweckregistern, bestehend aus den Registern R1 und R1+1. R1 muß geradzahlig sein, andernfalls erfolgt eine Programmunterbrechung wegen Adreßfehlers.

Der Dividend wird den Mehrzweckregistern R1 und R1+1 entnommen. Der Divisor ist beim Befehl DR im Mehrzweckregister R2 und beim Befehl D in dem mit D2(X2,B2) adressierten Hauptspeicherwort enthalten. Der Rest wird in das (geradzahlige) Register R1, der Quotient in das (ungeradzahlige) Register R1+1 gespeichert; sie überschreiben den Dividenden.

Der Dividend wird als 64 Bit lange Festpunktzahl mit dem Vorzeichen an der Bitstelle 0 von R1, der Divisor, Rest und Quotient werden als 32 Bit lange Festpunktzahlen mit Vorzeichen behandelt.

Das Vorzeichen des Quotienten wird nach den üblichen algebraischen Regeln ermittelt; der Rest hat stets das gleiche Vorzeichen wie der Dividend.

Wenn der Quotient zu groß ist zur Aufnahme im Register R1 oder der Divisor =0 ist, erfolgt eine Programmunterbrechung wegen Divisionsfehlers (auch wenn der Dividend =0 ist).

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	D : Lesezugriff auf Operand2 unmöglich
Adreßfehler	X'5C'	DR, D : R1 nicht geradzahlig
		D : D2(X2,B2) keine Wortgrenze
Divisionsfehler	X'68'	Divisor =0 oder Quotient zu groß.

Programmierhinweise

- Wenn R1=R2 ist, erfolgt in allen Fällen eine Programmunterbrechung wegen Divisionsfehlers.
- Die Maximalwerte für den Dividenden sind $+2^{62}+2^{31}-1$ und $-2^{62}+1$, nicht $+2^{63}-1$ und -2^{63} (siehe Beispiele).
- Man beachte, daß nach der Befehlsausführung der Rest *vor* dem Quotienten gespeichert ist (R1: Rest, R1+1: Quotient).

Beispiele

Die folgenden Werte von Dividend und Divisor ergeben die dargestellten Werte für Quotient und Rest:

Dividend	Divisor	Rest	Quotient
+500	+17	+7	+29
+500	-17	+7	-29
-500	+17	-7	-29
-500	-17	-7	+29
Grenzwerte:			
$+2^{62}+2^{31}-1$	$2^{31}-1$	$+2^{31}-2$	$+2^{31}-1$
$+2^{62}+2^{31}-1$	-2^{31}	-2^{31}	$+2^{31}-1$
$-2^{62}+2$	$+2^{31}-1$	$-2^{31}+2$	-2^{31}
$-2^{62}+1$	-2^{31}	$-2^{31}+1$	$+2^{31}-1$

Execute

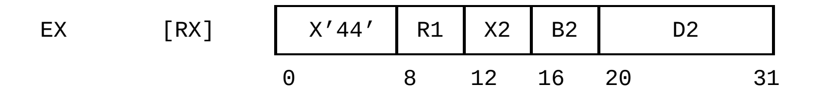
Funktion

Der Befehl EX führt einen anderen Befehl aus. Dieser Befehl kann zuvor modifiziert werden.
Die Anzeige wird nur verändert, wenn der ausgeführte Befehl sie verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	EX	R1,D2(X2,B2)	D2(X2,B2): Halbwortgrenze

Maschinenformat



Beschreibung

Der Befehl EX führt den mit D2(X2,B2) adressierten Befehl ("Ziel-Befehl") aus. Zuvor werden dessen Bitstellen 8 bis 15 mit den Bitstellen 24 bis 31 des Mehrzweckregisters R1 durch logisches ODER verknüpft. Die ODER-Verknüpfung verändert weder den Befehl selbst noch das Register R1, sondern beeinflußt nur die Befehlsinterpretation des Ziel-Befehls.

Wenn R1=0 ist, wird der Ziel-Befehl ohne vorige ODER-Verknüpfung ausgeführt.

Der Ziel-Befehl kann 2, 4 oder 6 Byte lang sein. Er wird so ausgeführt, als stünde er an der Speicherstelle des Befehls EX und hätte dessen Länge von 4 Byte. Wenn z.B. der Ziel-Befehl ein Befehl BALR ist, wird als "Befehlsfolgeadresse" die Folgeadresse des Befehls EX gespeichert (nicht die des BALR) und als ILC der Wert $(10)_2$ (nicht $(01)_2$).

Der Ziel-Befehl des Befehls EX darf nicht seinerseits ein EX sein, andernfalls erfolgt eine Programmunterbrechung wegen Adreßfehlers. Die mit D2(X2,B2) bestimmte Adresse muß geradzahlig sein, andernfalls erfolgt eine Programmunterbrechung ebenfalls wegen Adreßfehlers. Wenn der Ziel-Befehl kein korrekter Befehl des Befehlsvorrats ist, ist das Ergebnis des EX nicht vorhersehbar.

Anzeige

Die Anzeige wird in der Weise verändert, wie sie der Ziel-Befehl verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Adreßfehler	X’48’ X’5C’	Lesezugriff auf Ziel-Befehl unmöglich Ziel-Befehl ist seinerseits ein EX-Befehl oder D2(X2,B2) keine Halbwortgrenze.

Programmierhinweise

- Die durch den Befehl EX bewirkte ODER-Verknüpfung des zweiten Byte eines Ziel-Befehls ermöglicht die indirekte Bestimmung von dessen Längen-, Index-, Masken- oder Registerfeld oder von dessen zweitem Operationscode-Byte.
- Der Befehl EX hat besondere Bedeutung bei der Programmierung von sog. "read only"- (oder "reentrant")-Programmen, weil er den Ziel-Befehl nicht verändert (siehe Beispiel).
- Vorsicht ist geboten, wenn der Ziel-Befehl ein unterbrechbarer Befehl, z.B. der Befehl CLCL oder MVCL ist. In diesem Fall sollte nicht eines der Register X2 oder B2 des Befehls EX auch im Ziel-Befehl verwendet werden, weil dann nach EX deren Integrität nicht mehr sichergestellt ist. Auch sollte bei MVCL der Befehl EX nicht selbst im Empfangsfeld des Ziel-Befehls enthalten sein.
- Der Befehl EX ist sehr zeitaufwendig.

Beispiele

Beispiel 1

Die folgenden Befehle machen eine Zahl variabler Länge zu einer Zahl fester Länge und konvertieren sie dabei ins gepackte Format:

Name	Operation	Operanden
	LH	5,SLENGTH SLENGTH: Laenge der Zahl in SFIELD
	BCTR	5,0 minus 1
	EX	5,PACKINST
	.	
	.	
	.	
PACKINST	PACK	DFIELD,SFIELD(0) L1=L’DFIELD, L2=0
	.	

Die Länge der SFIELD-Zahl wird dem Halbwort SLENGTH entnommen und durch den Befehl EX in das L2-Feld des Befehls PACK ge-ODER-t, das deshalb $=0_{16}$ sein muß.

Zu beachten ist, daß die im Befehl PACK verwendete Länge gegenüber der wahren Länge um 1 vermindert sein muß, was hier durch den Befehl BCTR geschieht. Die Länge der DFIELD-Zahl ermittelt der Assembler [1] selbständig aus der Datenerklärung von DFIELD und reduziert sie auch selbständig um 1.

Beispiel 2

Die folgenden beiden Befehlsfolgen AAAA und BBBB bewirken das Gleiche, nämlich die Übertragung einer variablen Anzahl von Byte aus SFIELD in DFIELD:

Name	Operation	Operanden	
	.		
AAAA	LH	5,SLENGTH	SLENGTH: zu übertragende Bytezahl
	BCTR	5,0	minus 1
	EX	5,MOVEINST	
	.		
	.		
MOVEINST	MVC	DFIELD(0),SFIELD	

	.		
BBBB	LH	5,SLENGTH	SLENGTH: zu übertragende Bytezahl
	BCTR	5,0	
	STC	5,MOVEINST+1	Eintrag der um 1 verminderten
MOVEINST	MVC	DFIELD(0),SFIELD	Länge ins L-Feld eines MVC
	.		

Der Unterschied besteht darin, daß die Befehlsfolge AAAA "read only" bleibt, die Befehlsfolge BBBB jedoch nicht. Der Befehl EX führt den Befehl MVC zwar mit ge-ODER-tem Byte 1 aus, aber verändert ihn selbst nicht.
EX ist (nahezu) unverzichtbar bei Problemen, in denen der Programmtext konstant bleiben muß, aber dennoch dynamische Veränderungen in den Parametern einzelner Befehle erforderlich sind.

Insert Character

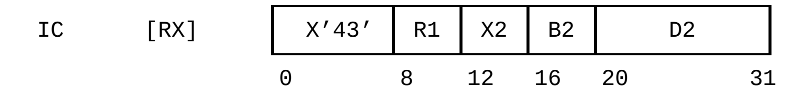
Funktion

Der Befehl IC überträgt ein Byte aus dem Speicher in das niedrigstwertige Byte eines Mehrzweckregisters.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	IC	R1,D2(X2,B2)	

Maschinenformat



Beschreibung

Das mit D2(X2,B2) adressierte Byte des Hauptspeichers wird in das Byte 3, d.h. in die Bitstellen 24 bis 31 des Mehrzweckregisters R1 geladen. Die Bitstellen 0 bis 23 von R1 bleiben unverändert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lesezugriff auf Operand2 unmöglich

Beispiel

Name	Operation	Operanden
	.	
	L	5,=XL4’ANNA’
	IC	5,=’E’ ’ANNE’ im Mehrzweckregister 5
*		Anzeige unverändert
	.	

Insert Characters under Mask

Funktion

Der Befehl ICM überträgt ein Zeichenfeld des Hauptspeichers in ausgewählte Byte eines Mehrzweckregisters.
Die Anzeige wird gemäß dem Wert des übertragenen Feldes gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	ICM	R1,M3,D2(B2)	B'0000' ≤ M3 ≤ B'1111'

Maschinenformat

ICM [RS]

X'BF'	R1	M3	B2	D2
0	8	12	16	20 31

Beschreibung

Die 4 Bit der "Maske" M3 (Direktoperand) entsprechen eins zu eins den 4 Byte des Mehrzweckregisters R1 (von links nach rechts sowohl in der Maske wie im Register). Diejenigen Byte in R1, denen Einsen in der Maske entsprechen, werden durch aufeinanderfolgende Byte des mit D2(B2) adressierten Hauptspeicherfelds ersetzt. Die Byte des Mehrzweckregisters, denen eine Null in der Maske entspricht, bleiben unverändert.

Ist die Maske $=0_{16}$ oder sind alle eingesetzten Byte $=00_{16}$, wird die Anzeige auf 0~Zero gesetzt. Andernfalls bestimmt das höchstwertige Bit des ersten eingesetzten Byte die Anzeige. Ist dieses Bit =1, so wird die Anzeige auf 1~Minus gesetzt, andernfalls auf 2~Plus.

Anzeige

0~Zero	Alle eingesetzten Byte sind $=00_{16}$ oder die Maske ist $=0_{16}$.
1~Minus	Das höchstwertige Bit des ersten eingesetzten Byte =1.
2~Plus	Das höchstwertige Bit des ersten eingesetzten Byte =0, aber mindestens ein weiteres Bit ist =1.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lesezugriff auf Operand2 unmöglich, auch wenn M3 = 0_{16} ist.

Programmierhinweise

- Die Länge in Byte des Hauptspeicherfelds ist gleich der Anzahl der Einsen in der Maske.
- Bei Verwendung einer Maske aus lauter Einsen (B’1111’) bestehen folgende Unterschiede zum Befehl L:
  - Das Hauptspeicherfeld braucht nicht an einer Wortgrenze ausgerichtet zu sein.
  - Die Anzeige wird gesetzt.
  - Der Befehl L hat RX- Format, der Befehl ICM RS-Format.
  - Der Befehl L ist schneller.

Beispiele

Folgende Beispiele ergeben:

Name	Operation	Operanden
	.	
Beispiel1	ICM	5,B’1111’,FBLENGTH
	.	
	.	
FBLENGTH	DC	C’ ’
	DC	FL3’20000’
	.	
	.	
Beispiel2	ICM	5,B’0001’,=X’00’
	.	

Im Beispiel1 werden 4 Byte in das Mehrzweckregister 5 eingesetzt: in Byte 0 ein Leerzeichen und in die Byte 1 bis 3 binär die Zahl 20000. Da das Leerzeichen das Coding X’40’ hat, also das erste eingesetzte Bit =0 ist, wird die Anzeige auf 2~Plus gesetzt.

Im Beispiel2 wird das niedrigstwertige Byte des Mehrzweckregisters 5 durch ein Byte des Hauptspeichers ersetzt. Im Gegensatz zum Befehl IC wird hier aber die Anzeige gesetzt, nämlich auf 0~Zero.

Insert Program Mask

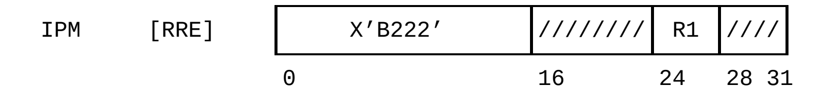
Funktion

Der Befehl IPM überträgt die momentanen Werte der Anzeige und der Programmaske in ein Mehrzweckregister.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	IPM	R1	

Maschinenformat



Beschreibung

Der momentane Wert der Anzeige (0_{10}, 1_{10}, 2_{10} oder 3_{10}) wird binär in die Bitstellen 2 und 3 und der momentane Wert der (4 Bit langen) Programmaske wird in die Bitstellen 4 bis 7 des Mehrzweckregisters R1 übertragen. Die Bitstellen 0 und 1 des Registers R1 werden auf 0 gesetzt; die Bitstellen 8 bis 31 des Registers R1 bleiben unverändert.

Die Bitstellen 16 bis 23 und 28 bis 31 des Befehls werden ignoriert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine.

Programmierhinweise

- Der Befehl IPM "entschädigt" dafür, daß es im 31-Bit-Adressierungsmodus nicht möglich ist, mit den Befehlen BALR oder BAL die Anzeige und die Programmaske zu lesen. Im 24-Bit-Adressierungsmodus ist dieses weiterhin möglich, aber auch da ist die Verwendung des Befehls IPM die bessere Lösung.
- Der Befehl IPM liefert *nicht* den "Instruction Length Code" (ILC), den die Befehle BALR und BAL (allerdings auch nur im 24-Bit-Adressierungsmodus) liefern. Auf ihn muß man im 31-Bit-Adressierungsmodus endgültig verzichten, (wenn er denn je gebraucht wird).
- Die Bit der Programmaske haben folgende Bedeutung:

Bit der Programmaske	Bitstelle in R1	Bedeutung
0	4	Festpunkt-Überlauf
1	5	Dezimal-Überlauf
2	6	Exponenten-Unterlauf
3	7	Signifikanz (Mantisse = 0)

Durch das BS2000 werden alle 4 Bit der Programmaske mit 1 vorbesetzt, so daß beim Auftreten des entsprechenden Ereignisses eine Programmunterbrechung eintritt. Durch den Befehl SPM kann aber ein Anwenderprogramm die Vorbesetzung ändern.

Beispiel

Name	Operation	Operanden
	.	
	ICM	15,B'1000',=X'3C'
	SPM	15 Anzeige 3~Overflow
	SLR	11,11 Anzeige 2~Plus
	IPM	11
	.	

Der Befehl setzt die Anzeige auf 3 und die Programmaske auf C_{16}. (Dadurch werden etwa folgende Programmunterbrechungen wegen Exponentunterlaufs und Signifikanz unterbunden, aber solche wegen Festpunkt- und Dezimal-Überlaufs zugelassen). Der Befehl SLR 11,11 läßt die Programmaske unverändert, aber setzt die Anzeige auf 2. Diesen Wert liest der Befehl IPM, so daß zuletzt das höchstwertige Byte des Registers 11 den Wert X'2C' (nicht X'3C') enthält.

Load

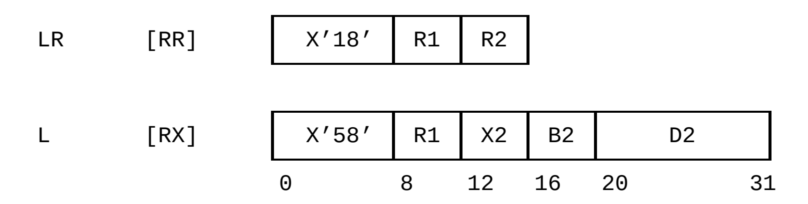
Funktion

Die Befehle LR und L übertragen eine 32 Bit lange Binärzahl aus einem Mehrzweckregister bzw. aus einem Wort des Hauptspeichers in ein Mehrzweckregister.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	LR L	R1,R2 R1,D2(X2,B2)	 D2(X2,B2): Wortgrenze

Maschinenformate



Beschreibung

Das mit D2(X2,B2) adressierte Wort des Hauptspeichers (L) bzw. der Inhalt des Mehrzweckregisters R2 (LR) wird in das Mehrzweckregister R1 übertragen.

Befehl	Operand1	Operand2
LR L	Register R1 Register R1	Register R2 mit D2(X2,B2) adressiertes Wort

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Adreßfehler	X’48’ X’5C’	L: Lesezugriff auf Operand2 unmöglich L: D2(X2,B2) keine Wortgrenze.

Load Address

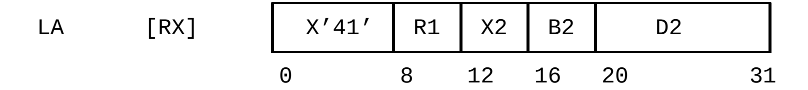
Funktion

Der Befehl LA lädt ein Mehrzweckregister mit einer Adresse.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	LA	R1,D2(X2,B2)	

Maschinenformat



Beschreibung

Die Adresse D2(X2,B2) wird in das Mehrzweckregister R1 geladen. Die Adresse wird logisch als Summe aus den Adressen in den Mehrzweckregistern X2 und B2 und dem Binärwert des 12 Bit langen D2-Feldes berechnet, wobei keine Vorzeichen berücksichtigt werden und etwaiger Übertrag über die höchstwertige Binärstelle ignoriert wird. Wenn X2=0 ist, wird der Inhalt des Registers X2, wenn B2=0 ist, wird der Inhalt des Registers B2 *nicht* mitsummiert. Im 24-Bit-Adressierungsmodus werden vom Inhalt der Mehrzweckregister B2 und X2 nur die niedrigstwertigen 24 Bit zur Summenbildung verwendet; die Summe wird in die Bitstellen 8 bis 31 des Mehrzweckregisters R1 eingetragen und die Bitstellen 0 bis 7 von R1 werden auf 0 gesetzt.
Im 31-Bit-Adressierungsmodus werden von B2 und X2 nur die niedrigstwertigen 31 Bit zur Summenbildung verwendet; die Summe wird in die Bitstellen 1 bis 31 des Mehrzweckregisters R1 eingetragen und das Bit 0 auf 0 gesetzt.

Es erfolgt kein Speicherzugriff auf die resultierende Adresse.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine.

Programmierhinweise

- Der Befehl LA ist oftmals ein kritischer Befehl bei der Portierung von Programmen aus dem 24-Bit-Adressierungsmodus in den 31-Bit-Adressierungsmodus. In älteren Programmen findet man nämlich nicht selten die höchstwertigen 8 Bit links von einer 24 Bit Adresse für zusätzliche Informationen, wie z.B. Anzeigen ausgenutzt und dann den Befehl LA zu dem Zweck eingesetzt, die höchstwertigen 8 Bit auf 0 zu setzen. Es empfiehlt sich, vor der Übertragung von Programmen aus einer 24-Bit-Umgebung in eine 31-Bit-Umgebung besonders alle die Adressen zu verfolgen, die von LA-Befehlen ausgehen.
- Der Befehl LA kann dazu verwendet werden, ein Mehrzweckregister um einen konstanten Wert zu erhöhen. Dazu trägt man diese Konstante als D2-Wert in den Befehl ein und setzt R1=B2 sowie X2=0, schreibt also z.B. LA 5,6(5), um das Mehrzweckregister 5 um 6 zu erhöhen. Man beachte jedoch, daß das Resultat des Befehls LA keine Festpunktzahl, sondern eine Adresse ist, die im 24-Bit-Adressierungsmodus eine andere Länge hat als im 31-Bit-Adressierungsmodus. Nur solange das Resultat kleiner als 16 MB ist, ist dieser Unterschied unerheblich.

Beispiel

Die "Gefährlichkeit" des Befehls LA sei an folgenden Befehlen erläutert:

Name	Operation	Operanden
	.	
A	CSECT	
A	AMODE	31
	.	
	.	
	LA	5,A
	L	15,=V(B)
	BASSM	14,15
*	.	
B	CSECT	
B	AMODE	24
	.	
	LA	5,1(5)
	BSM	0,14
	.	

Im Programmteil A wird eine 31 Bit lange Adresse im Register 5 erzeugt und im Programmteil B wird diese durch einen LA-Befehl um 1 erhöht. Da B im 24-Bit-Adressierungsmodus abläuft, wird auch nur eine 24 Bit lange Adresse erzeugt und an A zurückgegeben. Das Problem dabei ist, daß A im Adreßraum unterhalb von 16 MB richtig abläuft, oberhalb von 16 MB jedoch falsch.

Load Complement

Funktion

Der Befehl LCR überträgt das Zweierkomplement einer 32 Bit langen Festpunktzahl aus einem Mehrzweckregister in ein Mehrzweckregister.
Die Anzeige wird gemäß dem Wert der resultierenden Festpunktzahl gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	LCR	R1,R2	

Maschinenformat

		X'13'	R1	R2
LCR	[RR]	0	8	12 15

Beschreibung

Das Zweierkomplement der Festpunktzahl im Mehrzweckregister R2 wird in das Mehrzweckregister R1 übertragen.

Festpunkt-Überlauf entsteht, wenn die kleinste negative Zahl (-2^{31}) komplementiert werden soll; das Ergebnis in R1 ist dann wieder die kleinste negative Zahl und die Anzeige ist auf 3~Overflow gesetzt; außerdem erfolgt eine Programmunterbrechung, wenn das Bit für Festpunkt-Überlauf in der Programmaske =1 ist (BS2000-Standard).

Anzeige

0~Zero	Ergebnis = 0
1~Minus	Ergebnis < 0
2~Plus	Ergebnis > 0
3~Overflow	Festpunkt-Überlauf

Programmunterbrechungen

Art	Gewicht	Ursachen
Festpunkt-Überlauf	X'78'	R2-Inhalt = -2^{31}

Programmierhinweise

- R1 darf =R2 sein.
- Wenn der Inhalt von R2 =0 ist, wird auch der Inhalt von R1 (und die Anzeige) =0 gesetzt.

Beispiele

Name	Operation	Operanden	
	.		
Beispiel1	L	0,=F'-1'	Register 0: -1
	LCR	0,0	jetzt: +1, Anzeige: 2
	LCR	0,0	jetzt: -1, Anzeige: 1
	.		
	.		
Beispiel2	L	5,=F'-2147483648'	Register 5 : -2^{31}
	LCR	6,5	Register 6 : -2^{31}
	.		

Load Halfword

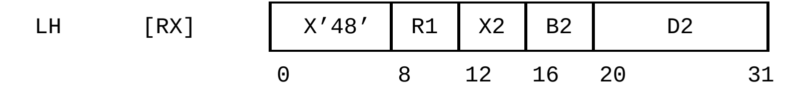
Funktion

Der Befehl LH überträgt ein Halbwort aus dem Hauptspeicher in die Byte 2 und 3 eines Mehrzweckregisters und füllt die Byte 0 und 1 mit dem Vorzeichenbit des Halbworts auf.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	LH	R1,D2(X2,B2)	D2(X2,B2): Halbwortgrenze

Maschinenformat



Beschreibung

Das mit D2(X2,B2) adressierte Halbwort wird in die Bitstellen 16 bis 31 des Mehrzweckregisters R1 übertragen. Die Bitstellen 0 bis 15 werden auf den Wert des höchstwertigen Bit des Halbworts gesetzt.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	Lesezugriff auf Operand2 unmöglich
Adreßfehler	X'5C'	D2(X2,B2) keine Halbwortgrenze.

Beispiel

Name	Operation	Operanden
	.	
	LH	0,=H’-1’
	CLM	0,B’1100’,=X’FFFF’ liefert Anzeige 0~Equal
	.	

Load Multiple

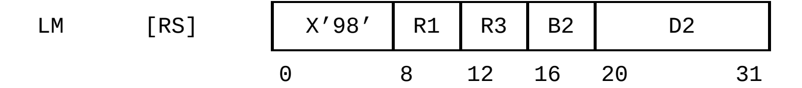
Funktion

Der Befehl LM lädt aus dem Hauptspeicher bis zu 16 aufeinanderfolgende Worte in aufeinanderfolgende Mehrzweckregister.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	LM	R1,R3,D2(B2)	D2(B2): Wortgrenze

Maschinenformat



Beschreibung

Die aufeinanderfolgenden Mehrzweckregister, beginnend mit R1 und endend mit R3, werden mit aufeinanderfolgenden Worten geladen, deren erstes mit D2(B2) adressiert ist.

Wenn R3 kleiner ist als R1, so wird von R1 aufwärts bis zum Mehrzweckregister 15 und vom Mehrzweckregister 0 bis zum und einschließlich von R3 geladen. Wenn R1=R3 ist, so wird nur ein Register (R1) geladen.

Befehl	Operand1	Operand2
LM	Inhalt von Register R1 bis R3	Mit D2(B2) adressierte Wortfolge; Wortanzahl =R3-R1+1, wenn R3≥R1 =R3-R1+17, wenn R3<R1

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Adreßfehler	X’48’ X’5C’	Lesezugriff auf Operand2 unmöglich D2(B2) keine Wortgrenze.

Beispiel

Name	Operation	Operanden
	.	
	LM	14,1,=A(ONE,TWO,THREE,FOUR)
*		Die Mehrzweckregister 14, 15, 0 und 1
*		werden mit 4 aufeinderfolgenden
*		Worten (hier: Adressen) geladen
	.	

Load Negative

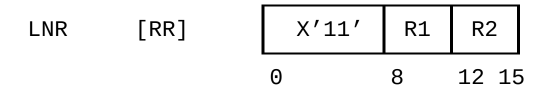
Funktion

Der Befehl LNR überträgt den negativen Wert einer 32 Bit langen Festpunktzahl aus einem Mehrzweckregister in ein Mehrzweckregister.
Die Anzeige wird gemäß dem Wert der übertragenen Festpunktzahl gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	LNR	R1,R2	

Maschinenformat



Beschreibung

Wenn die Festpunktzahl im Mehrzweckregister R2 positiv, d.h. wenn ihre Bitstelle 0 =0 ist, so wird ihr Zweierkomplement in das Mehrzweckregister R1 übertragen, andernfalls wird sie unverändert übertragen.

Wenn die zu übertragende Festpunktzahl =0 ist, wird auch die übertragene Zahl =0 gesetzt.

Anzeige

0~Zero	Ergebnis = 0 (R2 ist ebenfalls = 0)
1~Minus	Ergebnis < 0
2	Nicht verwendet.
3	Nicht verwendet.

Programmunterbrechungen

Keine.

Programmierhinweise

R1 darf =R2 sein.

Beispiele

Name	Operation	Operanden	
	.		
Beispiel1	L	0,=F’1’	Register 0 vorher: +1
	LNR	0,0	Register 0 nachher: -1
*			Anzeige: 1~Minus
	.		
	.		
Beispiel2	SLR	5,5	Register 5 : 0
	LNR	6,5	Register 6 : 0
*			Anzeige: 0~Zero
	.		

Load Positive

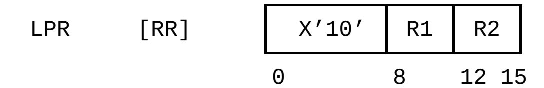
Funktion

Der Befehl LPR überträgt den Betrag einer 32 Bit langen Festpunktzahl aus einem Mehrzweckregister in ein Mehrzweckregister.
Die Anzeige wird gemäß dem Wert der übertragenen Festpunktzahl gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	LPR	R1,R2	

Maschinenformat



Beschreibung

Wenn die Festpunktzahl im Mehrzweckregister R2 negativ, d.h. ihre Bitstelle 0 den Wert =1 hat, so wird ihr Zweierkomplement in das Mehrzweckregister R1 übertragen, andernfalls wird sie unverändert übertragen.

Festpunkt-Überlauf entsteht, wenn die kleinste negative Zahl (-2^{31}) übertragen werden soll; das Ergebnis in R1 ist dann wieder die kleinste negative Zahl und die Anzeige ist auf 3~Overflow gesetzt; außerdem erfolgt eine Programmunterbrechung, wenn das Bit für Festpunkt-Überlauf in der Programmaske =1 ist (BS2000-Standard).

Anzeige

0~Zero	Ergebnis = 0
1	Nicht verwendet.
2~Plus	Ergebnis > 0
3~Overflow	Festpunkt-Überlauf

Programmunterbrechungen

Art	Gewicht	Ursachen
Festpunkt-Überlauf	X'78'	R2-Inhalt =-2^{31}

Programmierhinweise

R1 darf =R2 sein.

Beispiele

Name	Operation	Operanden	Bemerkungen
	.		
Beispiel1	L	0,=F'-1'	Register 0 vorher: -1
	LPR	0,0	Register 0 nachher: +1
*			Anzeige: 2~Plus
	.		
	.		
Beispiel2	L	5,=F'-2147483648'	Register 5 : -2^{31}
	LPR	6,5	Register 6 : -2^{31}
*			Anzeige: 3~Overflow und
*			ggf. Programmunterbrechung
	.		

Load and Test

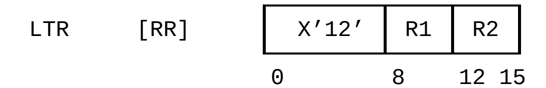
Funktion

Der Befehl LTR überträgt eine 32 Bit lange Festpunktzahl aus einem Mehrzweckregister in ein Mehrzweckregister.
Die Anzeige wird gemäß dem Wert der übertragenen Festpunktzahl gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	LTR	R1,R2	

Maschinenformat



Beschreibung

Die Festpunktzahl im Mehrzweckregister R2 wird unverändert in das Mehrzweckregister R1 übertragen und dabei auf ihren Wert getestet.
Festpunkt-Überlauf kann nicht auftreten.

Anzeige

0~Zero	Ergebnis = 0
1~Minus	Ergebnis < 0
2~Plus	Ergebnis > 0
3~Overflow	Nicht verwendet.

Programmunterbrechungen

Keine.

Programmierhinweise

- R1 darf =R2 sein.
- Der Befehl LTR leistet das gleiche wie der Befehl LR, aber setzt zusätzlich die Anzeige.

Multiply

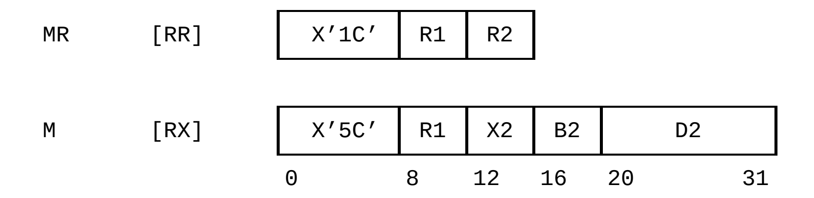
Funktion

Die Befehle MR und M multiplizieren zwei 32 Bit lange Festpunktzahlen vorzeichengerecht und erzeugen ein 64 Bit langes Produkt.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	MR M	R1,R2 R1,D2(X2,B2)	R1 geradzahlig R1 geradzahlig und D2(X2,B2): Wortgrenze

Maschinenformate



Beschreibung

Das R1-Feld der Befehle MR und M bestimt ein Paar von Mehrzweckregistern, bestehend aus den Registern R1 und R1+1. R1 muß geradzahlig sein, andernfalls erfolgt eine Programmunterbrechung wegen Adreßfehlers.

Der Multiplikand wird dem ungeradzahligen Mehrzweckregister R1+1 entnommen; der Inhalt des geradzahligen Registers R1 wird ignoriert. Der Multiplikator ist beim Befehl MR im Mehrzweckregister R2 und beim Befehl M in dem mit D2(X2,B2) adressierten Hauptspeicherwort enthalten. Das Produkt wird in die Register R1 und R1+1 gespeichert.

Der Multiplikand und Multiplikator werden als 32 Bit lange Festpunktzahlen mit Vorzeichen behandelt. Das resultierende Produkt ist eine 64 Bit lange Festpunktzahl mit dem Vorzeichen an der Bitstelle 0 des Mehrzweckregisters R1.

Das Vorzeichen des Produkts wird nach den üblichen algebraischen Regeln ermittelt.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	M : Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	MR, M : R1 nicht geradzahlig
		M : D2(X2,B2) keine Wortgrenze

Programmierhinweise

– Beim Befehl MR darf R2=R1 oder R2=R1+1 sein. Wenn R2=R1+1 ist, wird das Quadrat aus R2 ermittelt.

– Der größt- und kleinstmögliche Wert für das Produkt kann sich zu $+2^{62}$ bzw. $-2^{62}+2^{31}$ ergeben.

Beispiele

Die folgenden Werte von Multiplikand und Multiplikator ergeben die dargestellte Werte für das Produkt:

Multiplikand	Multiplikator	Produkt
+29	+17	+493
+29	-17	-493
Minimal- und Maximalwerte für das Produkt :		
$+2^{31}-1$	-2^{31}	$-2^{62}+2^{31}$
-2^{31}	-2^{31}	$+2^{62}$

Monitor Call

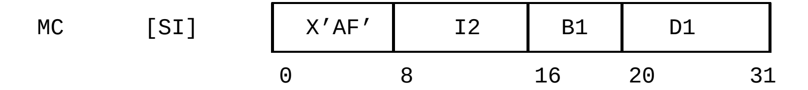
Funktion

Der Befehl MC erzeugt eine Programmunterbrechung wegen Monitoraufrufs.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	MC	D1(B1),I2	X'00' ≤ I2 ≤ X'0F'

Maschinenformat



Beschreibung

Es erfolgt eine Programmunterbrechung, wenn das Maskenbit für die Monitorklasse, die durch das I2-Feld des Befehls bestimmt ist, =1 gesetzt ist.

Der - je nach Adressierungsmodus entweder 24 Bit oder 31 Bit lange - Adreßwert D1(B1) dient als Argument für die Unterbrechungsroutine.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßfehler Monitor-Unterbrechung	X'5C'	I2 > 15 siehe Programmierhinweise.

Programmierhinweise

Der Befehl MC wird vom BS2000 nicht unterstützt. Wenn er dennoch aufgerufen wird, wirkt er wie eine NOP-Operation.

Multiply Halfword

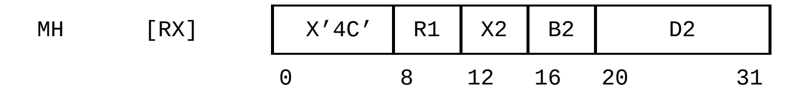
Funktion

Der Befehl MH multipliziert eine 32 Bit lange Festpunktzahl mit einer 16 Bit langen Festpunktzahl vorzeichengerecht und speichert die niedrigstwertigen 32 Binärstellen des Produkts.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	MH	R1,D2(X2,B2)	D2(X2,B2): Halbwortgrenze

Maschinenformat



Beschreibung

Der Multiplikand wird dem Mehrzweckregister R1, der Multiplikator dem mit D2(X2,B2) adressierten Halbwort des Hauptspeichers entnommen. Die niedrigstwertigen 32 Binärstellen des Produkts werden in das Mehrzweckregister R1 gespeichert und ersetzen den Multiplikanden.

Der Multiplikand wird als 32 Bit lange, der Multiplikator als 16 Bit lange Festpunktzahl mit dem Vorzeichen an der höchstwertigen Bitstelle behandelt. Das Produkt ist eine 48 Bit lange Festpunktzahl, von der nur die rechten 32 Binärstellen gespeichert werden. Die linken 16 Binärstellen einschließlich des Vorzeichenbit gehen verloren. Es erfolgt kein Test daraufhin, ob die verlorenen Binärstellen gleich sind dem Wert des höchstwertigen Bit des gespeicherten Ergebnisses.

Das Vorzeichen des Produkts wird nach den üblichen algebraischen Regeln ermittelt.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Adreßfehler	X’48’ X’5C’	Lesezugriff auf Operand2 unmöglich D2(X2,B2) keine Halbwortgrenze.

Programmierhinweise

– Da die höchstwertigen 16 Bit des echten Produkts verworfen werden, kann es geschehen, daß der Wert und/oder die Vorzeichenstelle des Ergebnisses verschieden sind vom Vorzeichen oder Wert des echten Produkts. Selbst wenn dies geschieht, wird es nicht angezeigt. Der Befehl MH sollte daher nur dann verwendet werden, wenn von Multiplikand und Multiplikator sicher bekannt ist, daß ihr Produkt im Bereich von -2^{31} und $+2^{31}-1$ liegt.

Beispiele

Die folgenden Werte von Multiplikand und Multiplikator ergeben die dargestellte Werte für das Ergebnis. Man beachte, daß das Ergebnis nur dann mit dem Produkt übereinstimmt, wenn das Produkt im Wertebereich von 32 Bit langen Festpunktzahlen liegt.

Multiplikand	Multiplikator	Ergebnis	Bemerkung
+29	+17	+493	arithmetisch korrekt
+29	-17	-493	arithmetisch korrekt
+131072	-32768	0	arithmetisch falsch, richtig wäre -4 295 464 296
+65538	+32767	+2 147 483 646	arithmetisch korrekt

Das letzte Beispiel zeigt eine arithmetische Grenze des Befehls MH: bereits die Werte +65539 und +32767 erzeugen ein arithmetisch unbrauchbares Resultat.

Move Characters

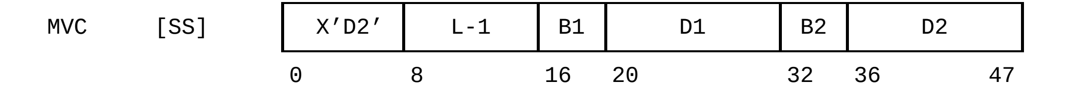
Funktion

Der Befehl MVC überträgt 1 bis 256 Byte aus einem Hauptspeicherbereich in einen anderen Hauptspeicherbereich.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	MVC	D1(L,B1),D2(B2)	1 ≤ L ≤ 256

Maschinenformat



Beschreibung

Das mit D2(B2) adressierte Zeichenfeld der Länge L Byte wird byteweise von links nach rechts in den mit D1(B1) adressierten Hauptspeicherbereich übertragen.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.

Programmierhinweise

- Die Operandenfelder dürfen sich überlappen.
- Man kann sich die Überlappungsmöglichkeit des MVC zunutze machen, um ein Feld zu "löschen", d.h. mit einem konstanten Bytewert zu füllen. Dazu speichert man diesen Bytewert in das Byte 0 (D1(B1)) des ersten Operanden (z.B. mit MVI) und führt danach einen MVC aus, dessen Operand1-Adresse =D1(B1)+1 und dessen Operand2-Adresse =D1(B1) ist. Dadurch wird das Byte 0 über den ersten Operanden "ausgebreitet" (siehe Beispiel).
- Für Feldlängen >256 Byte steht der Befehl MVCL zur Verfügung.

Beispiel

Name	Operation	Operanden
	. MVI MVC .	 FIELD,C' ' "Löschen" FIELD mit FIELD+1(L'FIELD-1),FIELD Leerzeichen

Move Long

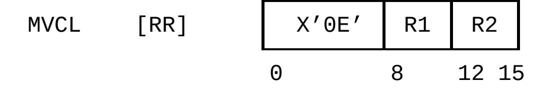
Funktion

Der Befehl MVCL überträgt den Inhalt eines Hauptspeicherbereichs von links nach rechts in einen anderen Hauptspeicherbereich und füllt diesen ggf. rechts mit Füllbytes auf. Die beiden Bereiche können bis zu 2^{24} Byte, d.h. bis zu 16 MB lang sein.
Die Anzeige wird gemäß dem Längenunterschied der beiden Bereiche gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	MVCL	R1,R2	R1 und R2 geradzahlig

Maschinenformat

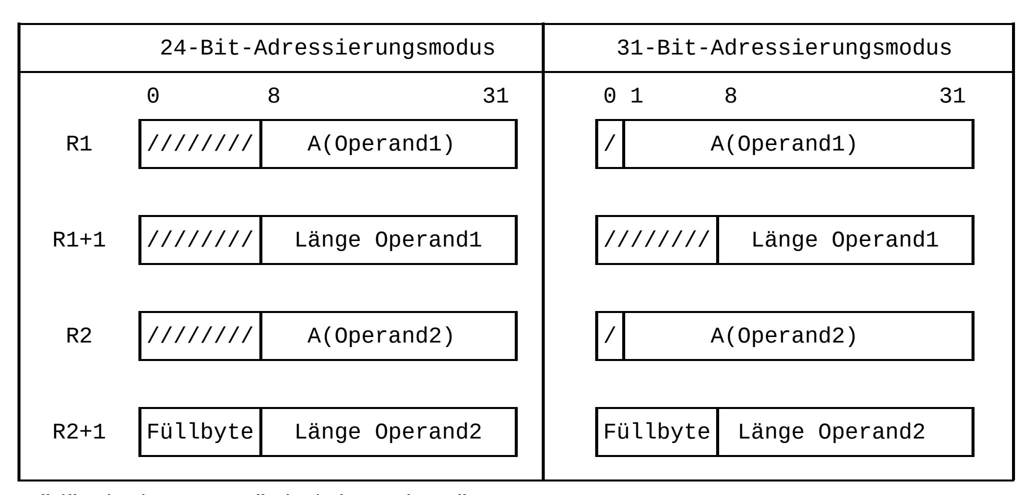


Beschreibung

Durch das R1-Feld des Befehls ist das Empfangsfeld, durch das R2-Feld ist das Sendefeld bestimmt. R1 und R2 bestimmen jeweils ein Paar von Mehrzweckregistern, bestehend aus den Registern R1 und R1+1 bzw. aus den Registern R2 und R2+1. R1 und R2 müssen beide geradzahlig sein, andernfalls wird nicht übertragen und es erfolgt eine Programmunterbrechung wegen Adreßfehlers.

Die Anfangsadressen des Empfangsfeldes und des Sendefeldes werden dem ersten, geradzahligen Register R1 bzw. R2 entnommen. Ihre Längen (in Byte) werden in den zweiten, ungeradzahligen Registern R1+1 bzw. R2+1 bestimmt. Das Register R2+1 enthält außerdem die Codierung des Füllbyte.

Die Adressendarstellung in R1 bzw. R2 ist abhängig vom Adressierungsmodus.
Es gilt folgende Zuordnung:



"/" bedeutet: "wird ignoriert"

Die Übertragung erfolgt byteweise von links nach rechts. Sie endet, wenn die durch R2+1 bestimmte Anzahl von Byte des Sendefelds in das Empfangsfeld übertragen ist. Wenn dann noch nicht die durch R1+1 bestimmte Länge des Empfangsfelds erreicht ist, wird das Empfangsfeld durch Füllbytes aufgefüllt, deren Codierung dem höchstwertigen Byte von R2+1 entnommen wird.

Die Übertragung wird nur durchgeführt, wenn sich das Empfangsfeld entweder nicht mit dem Sendefeld überlappt oder nur so überlappt, daß das Empfangsfeld nicht rechts vom Sendefeld beginnt. Für korrekte Überlappung muß gelten:

A(Empfangsfeld) ≤ A(Sendefeld)
oder A(Empfangsfeld) ≥ A(Sendefeld) + Min(L'Empfangsfeld, L'Sendefeld)

Bei inkorrekter (auch "destruktiv" genannter) Überlappung wird der Befehl nicht begonnen und die Anzeige auf 3~Overflow gesetzt.

Der Befehl MVCL ist hardwareseitig unterbrechbar. Bei einer Unterbrechung wird die bis dahin erreichte Übertragung in den Registerpaaren R1 und R2 festgehalten (durch Abspeicherung der hochgezählten Adressen und der heruntergezählten Längen). Nach der Unterbrechung wird dann die Übertragung an der unterbrochenen Stelle fortgesetzt.

Bei Befehlsende, also nach vollständiger Übertragung und ggf. erfolgter Auffüllung mit Füllbytes, werden in die Registerpaare R1 und R2 folgende Werte gespeichert: die Adressen in R1 und R2 sind um die Längenwerte in den Registern R1+1 bzw. R2+1 erhöht; die Register R1+1 und R2+1 enthalten in ihren niedrigstwertigen 3 Byte 00_{16}; die linken 1 oder 8 Bit vor den Adressen in R1 und R2 sind gleich 0 gesetzt, aber die linken 8 Bit von R1+1 und R2+1 (Füllbyte) sind unverändert.

Die Adreßfortschaltung im Sendefeld und im Empfangsfeld geschieht im 24-Bit-Adressierungsmodus modulo 2^{24} und im 31-Bit-Adressierungsmodus modulo 2^{31}. Demzufolge wird nach der Übertragung aus oder in das Byte mit der (virtuellen) Adresse 2^{24}-1 bzw. 2^{31}-1 als nächstes aus oder in das Byte mit der Adresse 0 übertragen (oder aufgefüllt), sofern die Operanden nicht vorher abgearbeitet sind.

Anzeige

0~Equal	Länge des Empfangsfelds = Länge des Sendefelds.
1~Low	Länge des Empfangsfelds < Länge des Sendefelds.
2~High	Länge des Empfangsfelds > Länge des Sendefelds.
3~Overflow	Empfangsfeld überlappt sich inkorrekt mit dem Sendefeld.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.
Adreßfehler	X’5C’	R1 oder R2 nicht geradzahlig

Programmierhinweise

- Wenn die Länge des Empfangsfelds =0 ist, wird nichts übertragen oder aufgefüllt und nur die Anzeige gesetzt.
- Wenn die Länge des Sendefelds =0 ist, wird das Empfangsfeld nur mit Füllbyte aufgefüllt. Damit läßt sich z.B. ein Empfangsfeld "löschen", d.h. mit einem konstanten Bytewert füllen.
- Man kann mit dem Befehl MVCL ein Empfangsfeld nicht auf die gleiche Weise löschen, wie das mit dem Befehl MVC möglich und üblich ist, nämlich durch "Ausbreiten" seines Byte 0. Wenn nämlich als Sendefeld die um Eins erhöhte Adresse des Emfangsfelds verwendet wird, erfolgt bei MVCL ein Befehlsabbruch wegen inkorrekter Überlappung.
- Die Prüfung auf inkorrekte Überlappung erfolgt (zu Befehlsbeginn) allein aufgrund der Daten in R1 und R2. Bei inkorrekter Überlappung wird der Befehl abgebrochen, wobei das Empfangsfeld unverändert ist. Es finden dann auch keine weiteren Prüfungen statt, so daß z.B. nicht entdeckt wird, ob alle Adressen von Sende- und Empfangsfeld auch vom Betriebssytem bereitgestellt sind.
- Eine andere Interpretation der Bedingungen für korrekte Überlappung ist folgende: Das Empfangsfeld muß so zum Sendefeld liegen, daß kein Byte zweimal übertragen werden muß.

- Wenn die Länge des Sendefelds =0 oder =1 beträgt, ist inkorrekte Überlappung nicht möglich.
- In Multiprozessor-Anwendungen muß ggf. folgendes beachtet werden: Da der Befehl hardwareseitig während seines Ablaufs unterbrochen werden kann, ist es möglich, daß das Empfangsfeld noch nicht vollständig gefüllt (oder gelöscht) ist, wenn eine andere Zentraleinheit schon auf es zugreift.
- Das Anwenderprogramm muß von sich aus sicherstellen, daß alle Adressen beider Operanden für das Programm ausschließlich in seinem eigenen Adreßraum liegen. Bei einer Befehlsbeendigung wegen Adreßumsetzungsfehlers kann die Übertragung bereits begonnen worden sein.
- Wegen der Unterbrechbarkeit des Befehls MVCL durch parallel arbeitende Zentraleinheiten sollte der Befehl MVCL, der die Übertragung auslöst, nicht selbst übertragen werden. Ebenso sollte ein Befehl EX, der einen Befehl MVCL ausführt, nicht mitübertragen werden.

Beispiel

Die folgenden Befehle übertragen 15000 Byte aus dem Bereich SF in den Bereich DF und füllen die anschließenden 5000 Byte des Bereichs DF mit dem Zeichen ’*’.

Name	Operation	Operanden	
	LM	4,5,=A(DF,20000)	R4,R5 : Operand1
	LM	10,11,=A(SF,15000)	R10,R11 : Operand2
	ICM	11,B’1000’,=’*’	Einsetzen Füllbyte in Byte 0
	MVCL	4,10	

Nach MVCL ist die Anzeige auf 2~High gesetzt (20000 > 15000). Die Register 4 bzw. 10 enthalten die Adressen A(DF+20000) bzw. A(SF+15000), die Register 5 und 11 enthalten in den rechten 3 Byte 00 00 00 und im linken Byte die Werte 00_{16} bzw. das Zeichen ’*’.
Voraussetzung für dieses Ergebnis ist, daß der Bereich DF entweder vor SF oder nach SF+14999 beginnt oder A(DF)=A(SF) ist (andernfalls ist die Anzeige 3~Overflow gesetzt und es hat keine Übertragung stattgefunden).

Move Immediate

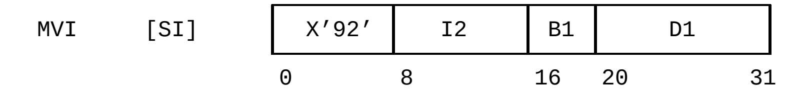
Funktion

Der Befehl MVI überträgt ein Byte (Direktoperand) in den Hauptspeicher.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	MVI	D1(B1),I2	X’00’ ≤ I2 ≤ X’FF’

Maschinenformat



Beschreibung

Der Direktoperand I2 ersetzt das mit D1(B1) adressierte Hauptspeicherbyte.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Schreibzugriff auf Operand1 unmöglich

Beispiel

Siehe das Beispiel unter MVC.

Move Numerics

Funktion

Der Befehl MVN überträgt die rechten Halbbyte eines Hauptspeicherbereichs in die rechten Halbbyte eines anderen Hauptspeicherbereichs.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	MVN	D1(L,B1),D2(B2)	1 ≤ L ≤ 256

Maschinenformat

		X'D1'	L-1	B1	D1	B2	D2
MVN	[SS]	0	8	16	20	32	36 47

Beschreibung

Die rechten Halbbyte des mit D2(B2) adressierten Zeichenfelds der Länge L Byte (d.h. die Ziffernteile) werden von links nach rechts in die rechten Halbbyte des mit D1(B1) adressierten Zeichenfelds übertragen; die linken Halbbyte des ersten Operanden bleiben unverändert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.

Programmierhinweise

- Die Operanden dürfen sich überlappen.
- Man kann sich die Überlappungsmöglichkeit des MVN zunutze machen, um die Ziffernteile eines Feld zu "löschen", d.h. mit einem konstanten Wert zu füllen: Dazu speichert man diesen Wert in das Byte 0 des ersten Operanden (z.B. mit OI) und führt danach einen MVN aus, dessen erste Operanden-Adresse =D1(B1)+1 und dessen zweite Operanden-Adresse =D1(B1) ist. Dadurch wird der rechte Teil des Byte 0 über den ersten Operanden "ausgebreitet".

Beispiel

Name	Operation	Operanden
	.	
	XC	DFIELD(3),DFIELD DFIELD : X’000000’
	MVN	DFIELD(3),=C’123’ DFIELD nachher: X’010203’
	.	

Move with Offset

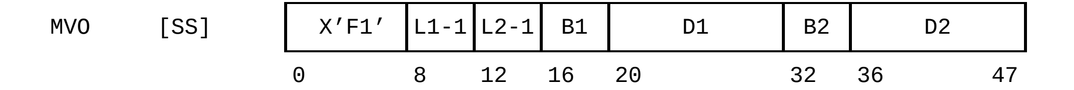
Funktion

Der Befehl MVO überträgt ein Zeichenfeld des Hauptspeichers um ein Halbbyte nach links versetzt in ein anderes Zeichenfeld.
Die Anzeige wird gemäß dem Vergleichsergebnis gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	MVO	D1(L1,B1),D2(L2,B2)	1 ≤ L1,L2 ≤ 16

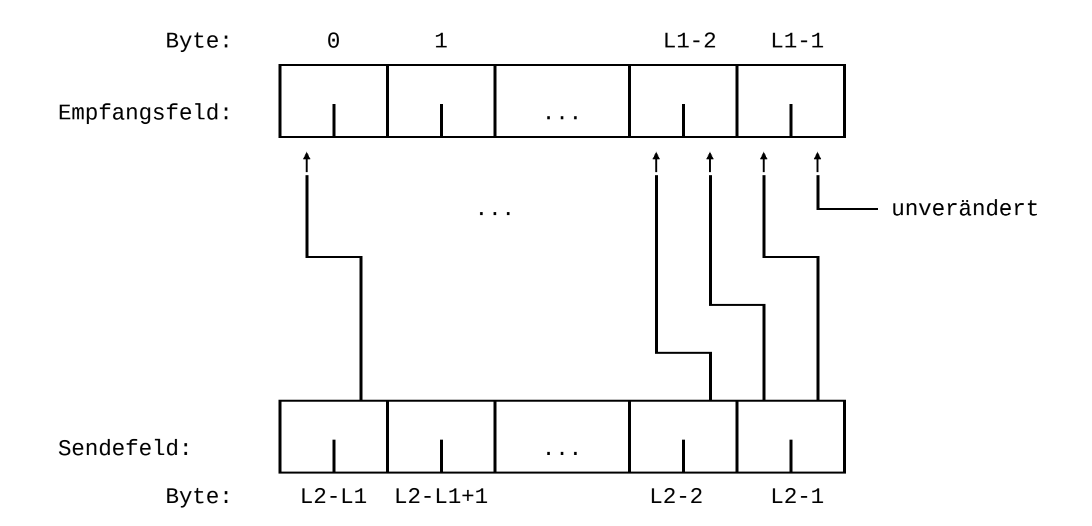
Maschinenformat



Beschreibung

Das mit D1(B1) adressierte Zeichenfeld im Hauptspeicher (Länge L1 Byte) ist das Empfangsfeld, das mit D2(B2) adressierte Zeichenfeld im Hauptspeicher (Länge L2 Byte) ist das Sendefeld.

Die Übertragung erfolgt von rechts nach links. Von jedem Byte des Sendefelds werden die rechten 4 Bit in die linken 4 Bit des gegenüberliegenden Byte des Empfangsfelds übertragen und die linken 4 Bit in die rechten 4 Bit des davorliegenden Byte im Empfangsfeld. Die rechten 4 Bit des niedrigstwertigen Byte des Empfangsfelds bleiben unverändert (siehe nachfolgende Darstellung).



In dieser Darstellung ist L2 ≥ L1 unterstellt; sonst ist das höchstwertige Byte des Empfangsfelds das Byte L1-L2-1 und das höchstwertige Byte des Sendefelds das Byte 0.

Das Empfangsfeld wird links mit 0_{16} aufgefüllt, wenn es länger ist als das Sendefeld; wenn das Empfangsfeld zu kurz ist, um alle Halbbyte des Sendefelds aufzunehmen, gehen die höchstwertigen Halbbyte des Sendefelds verloren.

Das Empfangsfeld darf sich mit dem Sendefeld überlappen. Die Übertragung wird so ausgeführt, als werde jedes Byte des Empfangsfelds unmittelbar dann gespeichert, wenn seine beiden Halbbyte ermittelt sind.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.

Programmierhinweise

Der Befehl MVO kann dazu benutzt werden, um eine gepackte Dezimalzahl um eine ungerade Anzahl von Dezimalstellen nach rechts zu verschieben (siehe Beispiel). Die Dezimalzahl wird aber nicht auf korrektes gepacktes Format geprüft.

Beispiel

Name	Operation	Operanden
	. MVO .	FIELD,FIELD(L’FIELD-2)

Obiger Befehl verschiebt den Inhalt von FIELD um 3 Halbbyte nach rechts, aber läßt das rechteste Byte von FIELD unverändert. Zum Beispiel ändert sich FIELD-vorher =X’ABCDEF’ zu FIELD-nachher =X’000ABF’.
Wenn der Inhalt von FIELD eine gepackte Dezimalzahl ist, ist das Ergebnis gleichbedeutend mit deren Ganzzahl-Division durch 1000.

Move Zones

Funktion

Der Befehl MVZ überträgt die linken Halbbyte eines Hauptspeicherbereichs in die linken Halbbyte eines anderen Hauptspeicherbereichs.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	MVZ	D1(L,B1),D2(B2)	1 ≤ L ≤ 256

Maschinenformat

MVZ [SS]

X’D3’	L-1	B1	D1	B2	D2
0	8	16	20	32	36 ... 47

Beschreibung

Die linken Halbbyte des mit D2(B2) adressierten Zeichenfelds der Länge L Byte (d.h. die Zonenteile) werden von links nach rechts in die linken Halbbyte des mit D1(B1) adressierten Zeichenfelds übertragen; die rechten Halbbyte des ersten Operanden bleiben unverändert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.

Programmierhinweise

- Die Operanden dürfen sich überlappen.
- Man kann sich die Überlappungsmöglichkeit des MVZ zunutze machen, um die Zonenteile eines Felds zu "löschen", d.h. mit einem konstanten Wert zu füllen: Dazu speichert man diesen Wert in das Byte 0 des ersten Operanden (z.B. mit NI und OI) und führt danach einen MVZ aus, dessen erste Operanden-Adresse =D1(B1)+1 und dessen zweite Operanden-Adresse =D1(B1) ist. Dadurch wird der linke Teil des Byte 0 über den ersten Operanden "ausgebreitet" (siehe Beispiel).

Beispiel

```
Name      Operation  Operanden
          .
          NI         DFIELD,X'0F'           Einsetzen Füll-Zone
          OI         DFIELD,X'C0'           in Byte 0
          MVZ        DFIELD+1(L'DFIELD-1),DFIELD
*                                           alle linken Halbbyte
*                                           werden =C16 gesetzt.
*                                           Die rechten Halbbyte
*                                           bleiben unverändert.
          .
```

AND

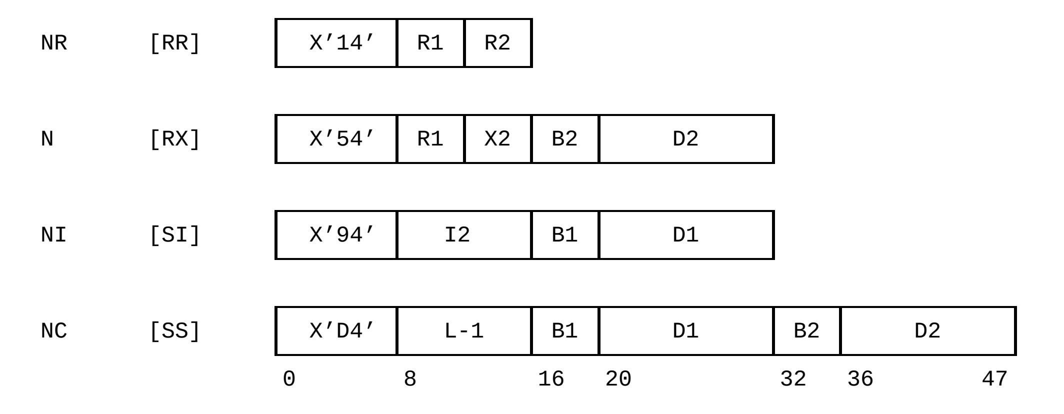
Funktion

Die Befehle NR, N, NI und NC bewirken bitweise die logische UND-Verknüpfung zweier Operanden.
Die Anzeige wird gemäß dem Wert des Ergebnisses gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	NR	R1,R2	
	N	R1,D2(X2,B2)	D2(X2,B2): Wortgrenze
	NI	D1(B1),I2	X’00’ ≤ I2 ≤ X’FF’
	NC	D1(L,B1),D2(B2)	1 ≤ L ≤ 256

Maschinenformate



Beschreibung

Die Bit des ersten Operanden werden gemäß der folgenden Tabelle durch die gegenüberliegenden Bit des zweiten Operanden verändert. Das Ergebnis ersetzt den ersten Operanden.

Tabelle der UND-Verknüpfungen

Bitwert im ersten Operanden	Bitwert im zweiten Operanden	Bitwert im Ergebnis
0	0	0
0	1	0
1	0	0
1	1	1

Operanden

Befehl	Operand1	Operand2
NR	Inhalt von Register R1	Inhalt von Register R2
N	Inhalt von Register R1	mit D2(X2,B2) adressiertes Wort
NI	mit D1(B1) adressiertes Byte	Direktoperand I2
NC	mit D1(B1) adressiertes Feld der Länge L Byte	mit D2(B2) adressiertes Feld der Länge L Byte

Anzeige

0~Zero	Ergebnis = 0
1~Not Zero	Ergebnis ≠ 0
2	Nicht verwendet.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	N: Lesezugriff auf Operand2 unmöglich NI: Lese-Schreibzugriff auf Operand1 unmöglich NC: Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	N: D2(X2,B2) keine Wortgrenze

Programmierhinweise

- Die AND-Befehle setzen im ersten Operanden alle Bitstellen auf 0, denen im zweiten Operanden eine Bitstelle mit dem Wert 0 gegenüberliegt und lassen die übrigen Bitstellen des ersten Operanden unverändert.
- Die Verarbeitung der Operanden erfolgt byteweise von links nach rechts.
- Bei NC dürfen sich die Operanden überlappen. Dabei werden allerdings i.a. frühere Byte-Operationen durch spätere wieder geändert.
- Wenn beim NR-Befehl R1=R2 ist, erfolgt keine Änderung des Inhalts von R1, aber die Anzeige wird gesetzt.
- Bei Anwendungen der Befehle NI und NC in Multiprozessor-Anlagen ist folgendes zu beachten:
  Speicherzugriffe des ersten Operanden der Befehle NI und NC bestehen aus dem Lesen eines Byte aus dem Speicher und dem anschließenden Abspeichern des veränderten Wertes in den Speicher. Diese Lese- und Abspeicherzugriffe auf ein einzelnes Byte geschehen nicht unbedingt sofort hintereinander, wenn ein weiterer Prozessor oder ein weiteres Programm (oder ein Kanalprogramm, Ein-/Ausgabe) versucht, diese Speicherstelle zu ändern. Eine sichere Methode zum Update eines gemeinsam benutzten Speicherworts ist im Anhang 7.6 sowie in den Programmierhinweisen der Befehle CS und CDS beschrieben.

Beispiel

Name	Operation	Operanden	
	.		
	NI	SEMAPHOR,X’F0’	Setzen der rechten vier Bit
*			des Byte SEMAPHOR auf 0_{16};
*			Die linken vier Bit bleiben
*			unverändert.
	.		

OR

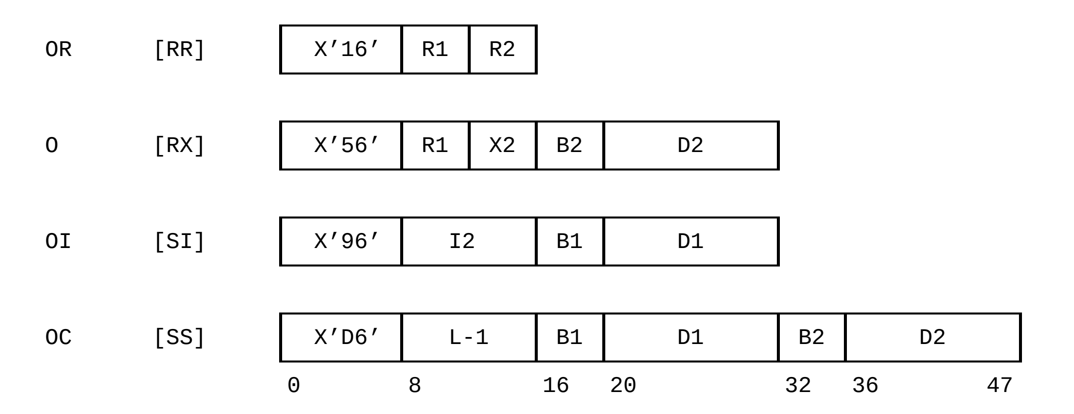
Funktion

Die Befehle OR, O, OI und OC bewirken bitweise die logische ODER-Verknüpfung zweier Operanden.
Die Anzeige wird gemäß dem Wert des Ergebnisses gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	OR	R1,R2	
	O	R1,D2(X2,B2)	D2(X2,B2): Wortgrenze
	OI	D1(B1),I2	X’00’ ≤ I2 ≤ X’FF’
	OC	D1(L,B1),D2(B2)	1 ≤ L ≤ 256

Maschinenformate



Beschreibung

Die Bit des ersten Operanden werden gemäß der folgenden Tabelle durch die gegenüberliegenden Bit des zweiten Operanden verändert. Das Ergebnis ersetzt den ersten Operanden.

Tabelle der ODER-Verknüpfungen

Bitwert im ersten Operanden	Bitwert im zweiten Operanden	Bitwert im Ergebnis
0	0	0
0	1	1
1	0	1
1	1	1

Operanden

Befehl	Operand1	Operand2
OR	Inhalt von Register R1	Inhalt von Register R2
O	Inhalt von Register R1	mit D2(X2,B2) adressiertes Wort
OI	mit D1(B1) adressiertes Byte	Direktoperand I2
OC	mit D1(B1) adressiertes Feld der Länge L Byte	mit D2(B2) adressiertes Feld der Länge L Byte

Anzeige

0~Zero	Ergebnis = 0
1~Not Zero	Ergebnis ≠ 0
2	Nicht verwendet.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	O: Lesezugriff auf Operand2 unmöglich OI: Lese-Schreibzugriff auf Operand1 unmöglich OC: Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	O: D2(X2,B2) keine Wortgrenze.

Programmierhinweise

- Die OR-Befehle setzen im ersten Operanden alle Bitstellen auf 1, denen im zweiten Operanden eine Bitstelle mit dem Wert 1 gegenüberliegt und lassen die übrigen Bitstellen des ersten Operanden unverändert.
- Die Verarbeitung der Operanden erfolgt byteweise von links nach rechts.
- Bei OC dürfen sich die Operanden überlappen. Dabei werden allerdings i.a. frühere Byte-Operationen durch spätere wieder geändert.
- Wenn beim OR-Befehl R1=R2 ist, also das Mehrzweckregister R1 mit sich selbst geODER-t wird, erfolgt keine Änderung des Inhalts von R1, aber die Anzeige wird gesetzt.
- Bei Anwendungen der Befehle OI und OC in Multiprozessor-Anlagen ist folgendes zu beachten:
  Speicherzugriffe des ersten Operanden der Befehle OI und OC bestehen aus dem Lesen eines Byte aus dem Speicher und dem anschließenden Abspeichern des veränderten Wertes in den Speicher. Diese Lese- und Abspeicherzugriffe auf ein einzelnes Byte geschehen nicht unbedingt sofort hintereinander, wenn ein weiterer Prozessor oder ein weiteres Programm (oder ein Kanalprogramm, Ein-/Ausgabe) versucht, diese Speicherstelle zu ändern. Eine sichere Methode zum Update eines gemeinsam benutzten Speicherworts ist im Anhang 7.6 sowie in den Programmierhinweisen der Befehle CS und CDS beschrieben.

Pack

Funktion

Der Befehl PACK erzeugt aus einer (entpackten) Dezimalzahl des Sendefelds eine gepackte Dezimalzahl im Empfangsfeld.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	PACK	D1(L1,B1),D2(L2,B2)	

Maschinenformat

PACK [SS]

X'F2'	L1-1	L2-1	B1	D1	B2	D2
0	8	12	16	20	32	36 47

Beschreibung

Durch D1(L1,B1) ist das Empfangsfeld, durch D2(L2,B2) ist das Sendefeld adressiert (1 ≤ L1,L2 ≤ 16). Die im Sendefeld enthaltene (entpackte) Dezimalzahl wird ins Empfangsfeld übertragen und dabei ins gepackte Format umgewandelt.

Das Sendefeld wird *nicht* darauf geprüft, ob es wirklich eine korrekte, entpackte Dezimalzahl enthält, sondern wird so behandelt, als enthalte es eine.

Beide Operanden werden von rechts nach links verarbeitet. Von jedem Byte des Sendefelds wird nur das rechte Halbbyte (der Ziffernteil) verwendet; jedes linke Halbbyte wird ignoriert mit Ausnahme des linken Halbbyte im niedrigstwertigen Byte des Sendefelds, das als Vorzeichen dient.

Das Vorzeichen und das rechte Halbbyte des niedrigstwertigen Byte des Sendefelds werden - in ihrer Reihenfolge vertauscht - in das niedrigstwertige Byte des Empfangsfelds übertragen. Alle weiteren rechten Halbbyte des Sendefelds werden aufeinanderfolgend in die weiteren Byte des Empfangsfelds übertragen und zwar immer zwei Halbbyte des Sendefelds in ein Byte des Empfangsfelds.

Wenn das Sendefeld ausgeschöpft ist, bevor das Empfangsfeld gefüllt ist, wenn also L2 < 2L1-1 ist, werden die höchstwertigen 2L1-L2-1 Halbbyte des Empfangsfelds mit 0_{16} besetzt. Wenn das Empfangsfeld zu kurz ist, um alle rechten Halbbyte des Sendefelds aufzunehmen, d.h., wenn 2L1 < L2+1 ist, werden die höchstwertigen L2-2L1+1 Byte des Sendefelds ignoriert.

Die beiden Operanden dürfen sich überlappen. Dabei verändert im allgemeinen eine spätere Byte-Operation eine frühere desselben Befehls. Der Befehl wird so ausgeführt, als werde jedes Byte des Empfangsfelds unmittelbar dann gespeichert, wenn die für es benötigten Halbbyte im Sendefeld gelesen sind.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.

Beispiele

Die folgenden Beispiele von PACK-Befehlen liefern folgende Ergebnisse:

DFIELD vorher	Beispiel-Befehl	DFIELD nachher
ohne Belang	PACK DFIELD(1),=Z’1’	P’1’
ohne Belang	PACK DFIELD(3),=Z’123’	P’00123’
ohne Belang	PACK DFIELD(1),=Z’123’	P’3’ plus Dezimal-Überlauf
X’89’	PACK DFIELD(1),DFIELD(1)	X’98’
X’23456789’	PACK DFIELD(2),DFIELD(4)	X’87986789’

Das letzte Beispiel zeigt einen (hoffentlich abschreckenden) Fall von Überlappung, bei dem sich während der Befehlsausführung das Sendefeld (!) selbst überschreibt.

Subtract

Funktion

Die Befehle SR und S subtrahieren zwei 32 Bit lange Festpunktzahlen vorzeichengerecht.
Die Anzeige wird gemäß dem Wert der Differenz gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	SR S	R1,R2 R1,D2(X2,B2)	 D2(X2,B2): Wortgrenze

Maschinenformate

SR	[RR]	X’1B’	R1	R2			
S	[RX]	X’5B’	R1	X2	B2	D2	
		0	8	12	16	20	31

Beschreibung

Durch den Befehl SR wird der Inhalt des Mehrzweckregisters R2, durch den Befehl S wird das mit D2(X2,B2) adressierte Wort des Hauptspeichers vorzeichengerecht vom Inhalt des Mehrzweckregisters R1 subtrahiert. Beide Operanden werden als 32 Bit lange Binärzahlen mit Vorzeichen (Festpunktzahlen) behandelt. Die Differenz ist ebenfalls eine 32 Bit lange Festpunktzahl und ersetzt den ursprünglichen Inhalt des Mehrzweckregisters R1.

Festpunkt-Überlauf entsteht, wenn die Differenz größer als 2^{31}-1 bzw. kleiner als -2^{31} wird. In diesem Fall ist das Ergebnis in R1 um 2^{32} zu klein bzw. zu groß; die Anzeige wird dann auf 3~Overflow gesetzt und es erfolgt eine Programmunterbrechung, wenn in der Programmaske das Bit für Festpunkt-Überlauf =1 ist (BS2000-Standard).

Anzeige

0~Zero	Differenz = 0
1~Minus	Differenz < 0
2~Plus	Differenz > 0
3~Overflow	Festpunkt-Überlauf

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	S: Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	S: D2(X2,B2) keine Wortgrenze.
Festpunkt-Überlauf	X’78’	Differenz > $+2^{31}-1$ oder < -2^{31}

Programmierhinweise

- Festpunkt-Überlauf entsteht dann, wenn ein Binärstellenüberlauf in die Vorzeichen-stelle ungleich ist dem Binärstellenüberlauf aus der Vorzeichenstelle. Im Register R1 hat dann das Resultat ein falsches Vorzeichen an der Bitstelle 0.
- SR mit R1=R2 "nullt" das Mehrzweckregister R1 und setzt die Anzeige auf 0~Zero. (SLR mit R1=R2 nullt ebenfalls das Mehrzweckregister R1, aber setzt die Anzeige auf 2~Plus).

Subtract Halfword

Funktion

Der Befehl SH subtrahiert eine 16 Bit lange Festpunktzahl von einer 32 Bit langen Festpunktzahl vorzeichengerecht.
Die Anzeige wird gemäß dem Wert der Differenz gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	SH	R1,D2(X2,B2)	D2(X2,B2): Halbwortgrenze

Maschinenformat

SH [RX]

X'4B'	R1	X2	B2	D2
0	8	12	16	20 … 31

Beschreibung

Das mit D2(X2,B2) adressierte Halbwort im Hauptspeicher wird vorzeichengerecht vom Inhalt des Mehrzweckregisters R1 subtrahiert. Der Register-Operand wird als 32 Bit lange, der Halbwort-Operand als 16 Bit lange Festpunktzahl behandelt, beide mit Vorzeichen. Die Differenz ist eine 32 Bit lange Festpunktzahl mit Vorzeichen und ersetzt den ursprünglichen Inhalt des Mehrzweckregisters R1.
Festpunkt-Überlauf entsteht, wenn die Differenz größer als 2^{31}-1 bzw. kleiner als -2^{31} wird. In diesem Fall ist das Ergebnis in R1 um 2^{32} zu klein bzw. zu groß; die Anzeige wird dann auf 3~Overflow gesetzt und es erfolgt eine Programmunterbrechung, wenn in der Programmaske das Bit für Festpunkt-Überlauf =1 ist (BS2000-Standard).

Anzeige

0~Zero	Differenz = 0
1~Minus	Differenz < 0
2~Plus	Differenz > 0
3~Overflow	Überlauf

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	D2(X2,B2) keine Halbwortgrenze.
Festpunkt-Überlauf	X’78’	Differenz > $+2^{31}-1$ oder < -2^{31}

Programmierhinweise

Festpunkt-Überlauf entsteht dann, wenn ein Binärstellenüberlauf in die Vorzeichenstelle ungleich ist dem Binärstellenüberlauf aus der Vorzeichenstelle. Im Register R1 hat dann das Resultat ein falsches Vorzeichen.

Subtract Logical

Funktion

Die Befehle SLR und SL subtrahieren zwei 32 Bit lange Binärzahlen logisch.
Die Anzeige wird gemäß dem Wert der Differenz gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	SLR SL	R1,R2 R1,D2(X2,B2)	 D2(X2,B2): Wortgrenze

Maschinenformate

SLR [RR]

X'1F'	R1	R2

SL [RX]

X'5F'	R1	X2	B2	D2
0	8	12	16	20 31

Beschreibung

Durch den Befehl SLR wird der Inhalt des Mehrzweckregisters R2, durch den Befehl SL wird das mit D2(X2,B2) adressierte Wort des Hauptspeichers logisch vom Inhalt des Mehrzweckregisters R1 subtrahiert.

Beide Operanden werden als 32 Bit lange Binärzahlen ohne Vorzeichen behandelt.

Die Differenz ist ebenfalls eine 32 Bit lange Binärzahl ohne Vorzeichen und ersetzt den ursprünglichen Inhalt des Mehrzweckregisters R1.

Alle 32 Bit beider Operanden sind an der Subtraktion beteiligt. Ein Übertrag über die Bitstelle 0 hinaus wird in der Anzeige dargestellt.

Anzeige

0	Nicht verwendet (siehe Programmierhinweise).
1~Minus	Differenz ≠0, kein Überlauf.
2~Plus	Differenz =0, Überlauf.
3~Overflow	Differenz ≠0, Überlauf.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Adreßfehler	X’48’ X’5C’	SL: Lesezugriff auf Operand2 unmöglich SL: D2(X2,B2) keine Wortgrenze.

Programmierhinweise

– Die logische Subtraktion besteht in der Addition des Einerkomplements des zweiten Operanden und zusätzlich der Addition von 1 zum Inhalt des Mehrzweckregisters R1, nicht in der Addition des Zweierkomplements. Deshalb entsteht in jedem Falle Überlauf, wenn der zweite Operand =0 ist, was durch die Anzeige-Werte 2 oder 3 angezeigt wird.

– Eine resultierende Differenz =0 erzeugt stets die Anzeige 2~Plus, nicht die Anzeige 0~Zero.

– Die logische Subtraktion erzeugt in allen Fällen das gleiche Ergebnis wie die arithmetische Subtraktion (durch SR, S oder SH), allerdings wird die Anzeige anders gesetzt und es erfolgt bei Überlauf keine Programmunterbrechung.

– Eine andere Interpretation der Werte der Anzeige ist die folgende:

0	Nicht verwendet.
1~Minus	Operand1 < Operand2
2~Plus	Operand1 = Operand2
3~Overflow	Operand1 > Operand2

– Der Befehl SL kann Anwendung finden bei der vorzeichengerechten Subtraktion von Festpunktzahlen, die länger als 32 Bit sind. Dabei verwendet man SL-Befehle zur Subtraktion der niederwertigen Wortpaare und benutzt den Befehl S zur Subtraktion des höchstwertigen Wortpaars; wenn nach der Subtraktion eines niedrigstwertigen Wortpaars die Anzeige auf 1~Minus gesetzt ist, d.h. das Operand1-Wort kleiner als das Operand2-Wort war, muß die Zahl +1 von der Differenz des nächsthöheren Wortpaars subtrahiert werden (siehe Beispiel2).

Beispiel

Name	Operation	Operanden	
	.		
Beispiel1	L	10,=F’1’	
	SL	10,=F’1’	Register 10: 0
*			aber Anzeige =2, nicht =0
*			siehe Programmierhinweise
	.		
	.		
Beispiel2	LM	0,1,FPNO1	Subtraktion von zwei 64 Bit
LOWSUB	SL	1,FPNO2+4	langen Festpunktzahlen
	BNM	HIGHSUB	
	SH	0,=H’1’	FPNO1+4 war < FPNO2+4
HIGHSUB	S	0,FPNO2	
	.		

Das Beispiel2 zeigt die vorzeichengerechte Subtraktion von zwei 64 Bit langen Festpunktzahlen FPNO1 und FPNO2: Das niederwertige Wortpaar wird mittels SL und das höherwertige Wortpaar wird mittels S subtrahiert. Im Falle von Unterlauf bei der Subtraktion des niederwertigen Wortpaars muß von der Differenz des höherwertigen Wortpaars noch +1 subtrahiert werden. Das Ergebnis steht im Beispielsfalle in den Mehrzweckregistern 0 und 1.

Shift Left Single

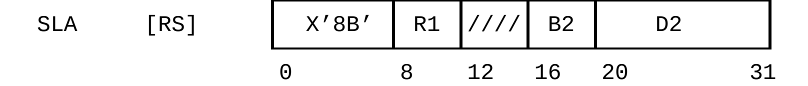
Funktion

Der Befehl SLA verschiebt eine 32 Bit lange Festpunktzahl in einem Mehrzweckregister vorzeichengerecht um eine angegebene Anzahl von Binärstellen nach links.
Die Anzeige wird gemäß dem Wert des Ergebnisses gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	SLA	R1,D2(B2)	
* oder auch:			
	SLA	R1,<anzahl>	

Maschinenformat



Beschreibung

Der Inhalt des Mehrzweckregisters R1 wird als 32 Bit lange Festpunktzahl mit dem Vorzeichen an der Bitstelle 0 behandelt.

Die durch D2(B2) bestimmte Adresse wird nicht als Datenadresse verwendet; stattdessen bilden die rechten 6 Bit dieser Adresse die Anzahl der Binärstellen, um die die Festpunktzahl nach links verschoben wird. Diese Anzahl liegt im Bereich zwischen 0 und 63_{10}. Die höherwertigen Binärstellen von D2(B2) werden ignoriert.

Bei der Linksverschiebung bleibt das Vorzeichen unverändert, nur die übrigen 31 Bitstellen werden verschoben. Rechts freiwerdende Bitstellen werden mit 0 gefüllt, links über die Bitstelle 1 von R1 hinausgeschobene Bitstellen gehen verloren.

Wenn ein oder mehrere vom Vorzeichenbit verschiedene Bit über die Bitstelle 1 des Registers R1 hinausgeschoben werden, tritt Festpunkt-Überlauf ein und die Anzeige wird auf 3~Overflow gesetzt. Wenn das Bit für Festpunkt-Überlauf in der Programmaske auf 1 gesetzt ist (BS2000-Standard), erfolgt außerdem eine Programmunterbrechung.

Die Bitstellen 12 bis 15 des Befehls werden ignoriert.

Anzeige

0~Zero	Verschobene Festpunktzahl = 0
1~Minus	Verschobene Festpunktzahl < 0 (Bit 0 von R1 =1)
2~Plus	Verschobene Festpunktzahl > 0 (Bit 0 von R1 =0)
3~Overflow	Es wurden ein oder mehrere vom Vorzeichenbit verschiedene Bit über die Bitstelle 1 von R1 hinausgeschoben.

Programmunterbrechungen

Art	Gewicht	Ursachen
Festpunkt-Überlauf	X'78'	siehe Anzeige 3~Overflow

Programmierhinweise

- Wenn B2=0 ist, bestimmt D2 allein die Verschiebeanzahl; in diesem Fall darf die Angabe von B2 im Assemblerformat entfallen.
- Wenn die Verschiebeanzahl =0 modulo 64 ist, wird das Register R1 nicht verändert, aber es wird die Anzeige gesetzt.
- Die Verschiebung um eine variable Anzahl von Bitstellen wird erreicht, indem man die Variable in das Mehrzweckregister B2 lädt.

Beispiele

Die folgenden Beispielbefehle ergeben:

Register 0 vorher	Beispielbefehl	Register 0 nachher	Anzeige
0...001 (+1)	SLA 0,1	0...010 (+2)	2
1...111 (-1)	SLA 0,1	1....10 (-2)	1
0...001 (+1)	SLA 0,30	010...0 (+2^{30})	2
0...001 (+1)	SLA 0,31	0.....0 (0)	3
10...00 (-2^{31})	SLA 0,1	10....0 (-2^{31})	3
10...00 (-2^{31})	SLA 0,128	10....0 (-2^{31})	1

Man beachte die beiden Fälle von Festpunkt-Überlauf (Anzeige =3): dieser Festpunkt-Überlauf kommt zustande, weil ein Bit ungleich dem Vorzeichen über die Bitstelle 1 hinausgeschoben wurde. Die Anzeige 3~Overflow zeigt in diesen Fällen an, daß das Ergebnis arithmetisch nicht korrekt ist.

Das letzte Beispiel zeigt einen Fall ohne Verschiebung: von der Verschiebezahl werden nur die niedrigstwertigen 6 Bit verwendet und diese ergeben bei der Zahl 128 den Wert 0. Zwar wird das Register 0 nicht verändert, aber es wird die Anzeige gesetzt.

Shift Left Double

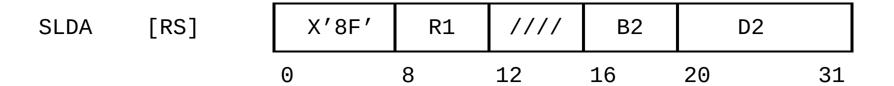
Funktion

Der Befehl SLDA verschiebt eine 64 Bit lange Festpunktzahl in einem Mehrzweckregister-Paar vorzeichengerecht um eine angegebene Anzahl von Binärstellen nach links. Die Anzeige wird gemäß dem Wert des Ergebnisses gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	SLDA	R1,D2(B2)	R1 geradzahlig
* oder auch:	SLDA	R1,<anzahl>	R1 geradzahlig

Maschinenformat



Beschreibung

Das R1-Feld des Befehls bestimmt ein Paar von Mehrzweckregistern, bestehend aus den Registern R1 und R1+1; R1 muß geradzahlig sein, andernfalls erfolgt eine Programmunterbrechung wegen Adreßfehlers.

Die Bitstellen 12 bis 15 des Befehls werden ignoriert.

Die durch D2(B2) bestimmte Adresse wird nicht als Datenadresse verwendet; stattdessen bilden die rechten 6 Bit dieser Adresse die Anzahl der Binärstellen, um die die Festpunktzahl nach links verschoben wird. Diese Anzahl liegt im Bereich zwischen 0 und 63_{10}. Die höherwertigen Binärstellen von D2(B2) werden ignoriert.

Der Inhalt des Mehrzweckregisterpaars R1 und R1+1 wird als 64 lange Festpunktzahl mit Vorzeichen behandelt. Das Vorzeichen an der Bitstelle 0 des (geradzahligen) Registers R1 bleibt unverändert, aber alle übrigen 63 Bitstellen werden verschoben. Rechts freiwerdene Bitstellen werden mit 0 gefüllt, links über die Bitstelle 1 von R1 hinausgeschobene Bitstellen gehen verloren.

Wenn ein oder mehrere vom Vorzeichenbit verschiedene Bit über die Bitstelle 1 des Registers R1 hinausgeschoben werden, tritt Festpunkt-Überlauf ein und die Anzeige wird auf 3~Overflow gesetzt. Wenn das Bit für Festpunkt-Überlauf in der Programmaske auf 1 gesetzt ist (BS2000-Standard), erfolgt außerdem eine Programmunterbrechung.

Anzeige

0~Zero	Verschobene Festpunktzahl = 0
1~Minus	Verschobene Festpunktzahl < 0 (Bit 0 von R1 =1)
2~Plus	Verschobene Festpunktzahl > 0 (Bit 0 von R1 =0)
3~Overflow	Es wurden ein oder mehrere vom Vorzeichenbit verschiedene Bit über die Bitstelle 1 von R1 hinausgeschoben.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßfehler Festpunkt-Überlauf	X’5C’ X’78’	R1 nicht geradzahlig. siehe Anzeige 3~Overflow

Programmierhinweise

- Wenn B2=0 ist, bestimmt D2 allein die Verschiebeanzahl; in diesem Fall darf die Angabe von B2 im Assemblerformat entfallen.
- Wenn die Verschiebeanzahl =0 modulo 64 ist, werden R1 und R1+1 nicht verändert, aber es wird die Anzeige gesetzt.
- Die Verschiebung um eine variable Anzahl von Bitstellen wird erreicht, indem man die Variable in das Mehrzweckregister B2 lädt.

Beispiele

Die folgenden Beispielbefehle ergeben:

Register 0,1 vorher	Beispielbefehl	Register 0,1 nachher	Anzeige
0...0 0..01 (+1)	SLDA 0,1	00...0 0..010 (+2)	2
1...1 1...1 (-1)	SLDA 0,1	11...1 1...10 (-2)	1
01..1 1...1 ($+2^{63}-1$)	SLDA 0,1	01...1 1...10 ($+2^{63}-2$)	3
0...0 1...1 ($+2^{32}-1$)	SLDA 0,31	01...1 10...0 ($+2^{63}-2^{31}$)	2
10..0 0...0 (-2^{63})	SLDA 0,1	10...0 0....0 (-2^{63})	3
10..0 0...0 (-2^{63})	SLDA 0,64	10...0 0....0 (-2^{63})	1

Man beachte die beiden Fälle von Festpunkt-Überlauf (Anzeige=3); dieser Festpunkt-Überlauf kommt zustande, weil ein Bit ungleich dem Vorzeichenbit über die Bitstelle 1 von Register 0 hinausgeschoben wurde. Die Anzeige 3~Overflow zeigt an, daß das Ergebnis arithmetisch nicht korrekt ist.
Das letzte Beispiel zeigt einen Fall ohne Verschiebung: von der Verschiebezahl werden nur die niedrigstwertigen 6 Bit verwendet und diese ergeben bei der Zahl 64 den Wert 0. Zwar werden die Register nicht verändert, aber die Anzeige wird gesetzt.

Shift Left Double Logical

Funktion

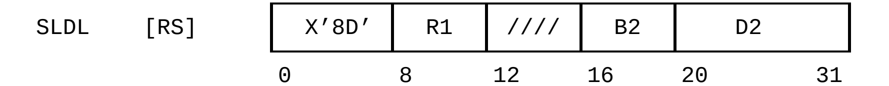
Der Befehl SLDL verschiebt eine 64 Bit lange Binärzahl in einem Mehrzweckregister-Paar um eine angegebene Anzahl von Binärstellen logisch nach links.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* oder auch:	SLDL	R1,D2(B2)	R1 geradzahlig
	SLDL	R1,<anzahl>	R1 geradzahlig

Maschinenformat

SLDL [RS]



Beschreibung

Das R1-Feld des Befehls bestimmt ein Paar von Mehrzweckregistern, bestehend aus den Registern R1 und R1+1; R1 muß geradzahlig sein, andernfalls erfolgt eine Programmunterbrechung wegen Adreßfehlers.

Die Bitstellen 12 bis 15 des Befehls werden ignoriert.

Die durch D2(B2) bestimmte Adresse wird nicht als Datenadresse verwendet; stattdessen bestimmen die rechten 6 Bit dieser Adresse die Anzahl der Binärstellen, um die die Binärzahl nach links verschoben wird. Diese Anzahl liegt im Bereich zwischen 0 und 63_{10}. Die höherwertigen Binärstellen von D2(B2) werden ignoriert.

Der Inhalt der beiden Register R1 und R1+1 wird als 64 Bit lange Binärzahl ohne Vorzeichen behandelt. Alle 64 Binärstellen dieser Zahl werden verschoben. Rechts freiwerdende Bitstellen werden mit 0 gefüllt, links über die Bitstelle 0 hinausgeschobene Bitstellen gehen verloren.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßfehler	X’5C’	R1 nicht geradzahlig.

Programmierhinweise

- Wenn B2=0 ist, bestimmt D2 allein die Verschiebeanzahl; in diesem Fall darf die Angabe von B2 im Assemblerformat entfallen.
- Wenn die Verschiebeanzahl =0 modulo 64 ist, werden R1 und R1+1 (und auch die Anzeige) nicht verändert.
- Die Verschiebung um eine variable Anzahl von Bitstellen wird erreicht, indem man die Variable in das Mehrzweckregister B2 lädt.

Beispiele

Die folgenden Beispielbefehle ergeben:

Register 0,1 vorher	Beispielbefehl	Register 0,1 nachher	Anzeige
0...0 0..01	SLDL 0,1	00...0 0..010	unverändert
1...1 1...1	SLDL 0,1	11...1 1...10	unverändert
01..1 1...1	SLDL 0,1	1....1 1...10	unverändert
0...0 1...1	SLDL 0,31	01...1 10...0	unverändert
10..0 0...0	SLDL 0,1	0....0 0....0	unverändert
10..0 0...0	SLDL 0,64	10...0 0....0	unverändert

Der Leser möge diese Beispiele mit denen bei der Beschreibung des Befehls SLDA vergleichen. Hier tritt in keinem Fall Festpunkt-Überlauf auf.
Das letzte Beispiel zeigt (wie bei SLDA) einen Fall ohne Verschiebung: von der Verschiebezahl werden nur die niedrigstwertigen 6 Bit verwendet und diese ergeben bei der Zahl 64 den Wert 0. Die Registerinhalte und die Anzeige bleiben unverändert.

Shift Left Single Logical

Funktion

Der Befehl SLL verschiebt eine 32 Bit lange Binärzahl in einem Mehrzweckregister um eine angegebene Anzahl von Binärstellen logisch nach links.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	SLL	R1,D2(B2)	
* oder auch:			
	SLL	R1,<anzahl>	

Maschinenformat

SLL [RS]

X’89’	R1	////	B2	D2
0	8	12	16	20 31

Beschreibung

Der Inhalt des Mehrzweckregisters R1 wird als 32 Bit lange Binärzahl ohne Vorzeichen behandelt.

Die durch D2(B2) bestimmte Adresse wird nicht als Datenadresse verwendet; stattdessen bilden die rechten 6 Bit dieser Adresse die Anzahl der Binärstellen, um die die Binärzahl nach links verschoben wird. Diese Anzahl liegt im Bereich zwischen 0 und 63_{10}. Die höherwertigen Binärstellen von D2(B2) werden ignoriert.

Bei der Linksverschiebung werden alle 32 Bitstellen verschoben. Rechts freiwerdende Bitstellen werden mit 0 gefüllt, links über die Bitstelle 0 hinausgeschobene Binärstellen gehen verloren.

Die Bitstellen 12 bis 15 des Befehls werden ignoriert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine.

Programmierhinweise

– Wenn B2=0 ist, bestimmt D2 allein die Verschiebeanzahl; in diesem Fall darf die Angabe von B2 im Assemblerformat entfallen.

– Wenn die Verschiebeanzahl =0 modulo 64 ist, bleibt das Register R1 (und die Anzeige) unverändert.

– Die Verschiebung um eine variable Anzahl von Bitstellen wird erreicht, indem man die Variable in das Mehrzweckregister B2 lädt.

Beispiele

Die folgenden Beispielbefehle ergeben:

Register 0 vorher	Beispielbefehl	Register 0 nachher	Anzeige
00....01	SLL 0,1	00....010	unverändert
11....11	SLL 0,1	11....110	unverändert
10....00	SLL 0,1	00....000	unverändert
10....00	SLL 0,64	10....000	unverändert

Das letzte Beispiel zeigt einen Fall ohne Verschiebung: von der Verschiebezahl werden nur die niedrigstwertigen 6 Bit verwendet und diese ergeben bei der Zahl 64 den Wert 0. Weder werden die Register noch wird die Anzeige verändert.

Set Program Mask

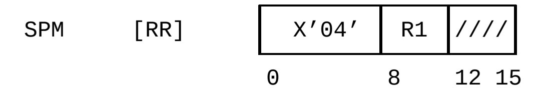
Funktion

Der Befehl SPM setzt die Programmaske und die Anzeige auf angegebene Werte.
Die Anzeige wird gemäß dem neuen Wert der Anzeige gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	SPM	R1	

Maschinenformat



Beschreibung

Die Bitstellen 2 und 3 des Mehrzweckregisters R1 ersetzen den (bisherigen) Wert der Anzeige und die Bitstellen 4 bis 7 von R1 ersetzen den (bisherigen) Wert der Programmaske.

Die Bitstellen 0 und 1 sowie 8 bis 31 des Mehrzweckregisters R1 werden ignoriert.
Ebenso werden die Bitstellen 12 bis 15 des Befehls SPM ignoriert.

Anzeige

0~Equal	Die Bitstellen 2 und 3 von R1 sind $=00_2$.
1~Low	Die Bitstellen 2 und 3 von R1 sind $=01_2$.
2~High	Die Bitstellen 2 und 3 von R1 sind $=10_2$.
3~Overflow	Die Bitstellen 2 und 3 von R1 sind $=11_2$.

Programmunterbrechungen

Keine.

Programmierhinweise

- Der Befehl SPM ermöglicht es, die vom BS2000 mit $(1111)_2$ vorbesetzte Programmaske zu ändern. Dadurch ist es dem Anwenderprogramm möglich, beim Eintreten der nachfolgend genannten vier Arten von Ereignissen die sonst stattfindende Programmunterbrechung zu unterbinden. Dafür kann es durchaus gute Gründe geben. Zum Beispiel wird man regelmäßig in Programmen, die intensiv Gleitpunktoperationen durchführen, die unvermeidlichen Programmunterbrechungen wegen Signifikanz auf diese Weise ausschalten.
  Es ist aber guter Programmierstil, anschließend die Programmaske wieder auf ihren ursprünglichen Zustand zu setzen.

- Die Bit der Programmaske haben (von links nach rechts) folgende Bedeutung:

Bit der Programmaske	Bitstelle in R1	Bedeutung
0	4	Festpunkt-Überlauf
1	5	Dezimal-Überlauf
2	6	Exponenten-Unterlauf
3	7	Signifikanz

Beispiel

```
Name      | Operation | Operanden
----------+-----------+------------------------------------------------------
          | .         |
*         |           |                        ┌──────  Exponenten-Unterlauf
*         |           |                        │
*         |           |                        ▼
          | ICM       | 15,B'1000',=B'00111100'
          | SPM       | 15                      ▲
*         |           |                         │
*         |           |                         └──────  Signifikanz
          | .         |
```

Nach Ausführung des SPM ist die Anzeige auf 3~Overflow gesetzt, und die Programmunterbrechungen Exponenten-Unterlauf und Signifikanz sind unterbunden.

Shift Right Single

Funktion

Der Befehl SRA verschiebt eine 32 Bit lange Festpunktzahl in einem Mehrzweckregister vorzeichengerecht um eine angegebene Anzahl von Binärstellen nach rechts.
Die Anzeige wird gemäß dem Wert des Ergebnisses gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	SRA	R1,D2(B2)	
* oder auch:			
	SRA	R1,<anzahl>	

Maschinenformat

		X'8A'	R1	////	B2	D2
SRA	[RS]	0	8	12	16	20 31

Beschreibung

Der Inhalt des Mehrzweckregisters R1 wird als 32 Bit lange Festpunktzahl mit dem Vorzeichen an der Bitstelle 0 behandelt.

Die durch D2(B2) bestimmte Adresse wird nicht als Datenadresse verwendet; stattdessen bilden die rechten 6 Bit dieser Adresse die Anzahl der Binärstellen, um die die Festpunktzahl nach rechts verschoben wird. Diese Anzahl liegt im Bereich zwischen 0 und 63_{10}. Die höherwertigen Binärstellen von D2(B2) werden ignoriert.

Bei der Rechtsverschiebung bleibt das Vorzeichen unverändert, nur die übrigen 31 Bitstellen werden verschoben. Links freiwerdende Bitstellen werden mit dem Wert des Vorzeichenbit gefüllt, rechts über die Bitstelle 31 von R1 hinausgeschobene Binärstellen gehen verloren.

Die Bitstellen 12 bis 15 des Befehls werden ignoriert.

Anzeige

0~Zero	Verschobene Festpunktzahl = 0
1~Minus	Verschobene Festpunktzahl < 0 (Bit 0 von R1 =1)
2~Plus	Verschobene Festpunktzahl > 0 (Bit 0 von R1 =0)
3	Nicht verwendet.

Programmunterbrechungen

Keine.

Programmierhinweise

- Wenn B2=0 ist, bestimmt D2 allein die Verschiebeanzahl; in diesem Fall darf die Angabe von B2 im Assemblerformat entfallen.
- Wenn die Verschiebeanzahl =0 modulo 64 ist, bleibt R1 unverändert, aber die Anzeige wird gesetzt.
- Die Rechtsverschiebung von negativen Festpunktzahlen führt zur "Abwärtsrundung" auf die nächstniedrige negative Ganzzahl. So ergibt z.B. die Zahl -1 bei jedweder Rechtsverschiebung wieder die Zahl -1 oder es ergibt die Zahl -5, wenn sie um 2 Binärstellen nach rechts geschoben, also durch 4 dividiert wird, die Zahl -2 (und nicht -1). Siehe dazu die Beispiele.

Beispiele

Die folgenden Beispielbefehle ergeben:

Register 0 vorher	Beispielbefehl	Register 0 nachher	Anzeige
0..0101 (+5)	SRA 0,2	0..0001 (+1)	2
1..1011 (-5)	SRA 0,2	1..1110 (-2)	1
0..0101 (+5)	SRA 0,3	0..0000 (0)	0
1..1011 (-5)	SRA 0,3	1..1111 (-1)	1
0..0001 (+1)	SRA 0,31	0..0000 (0)	0
1..1111 (-1)	SRA 0,31	1..1111 (-1)	1
01..111 ($+2^{31}-1$)	SRA 0,31	0..0000 (0)	0
10..001 ($-2^{31}+1$)	SRA 0,31	1..1111 (-1)	1

In den Beispielen ist jeweils die Rechtsverschiebung einer positiven Festpunktzahl der Rechtsverschiebung des negativen Pendants gegenübergestellt, um die vielleicht ungewohnten Unterschiede zu verdeutlichen.

Shift Right Double

Funktion

Der Befehl SRDA verschiebt eine 64 Bit lange Festpunktzahl in einem Mehrzweckregister-Paar vorzeichengerecht um eine angegebene Anzahl von Binärstellen nach rechts. Die Anzeige wird gemäß dem Wert des Ergebnisses gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	SRDA	R1,D2(B2)	R1 geradzahlig
* oder auch:			
	SRDA	R1,<anzahl>	R1 geradzahlig

Maschinenformat

SRDA [RS]

X'8E'	R1	////	B2	D2
0	8	12	16	20 31

Beschreibung

Das R1-Feld des Befehls bestimmt ein Paar von Mehrzweckregistern, bestehend aus den Registern R1 und R1+1; R1 muß geradzahlig sein, sonst erfolgt eine Programmunterbrechung wegen Adreßfehlers. Die Bitstellen 12 bis 15 des Befehls werden ignoriert.
Die durch D2(B2) bestimmte Adresse wird nicht als Datenadresse verwendet; stattdessen bilden die rechten 6 Bit dieser Adresse die Anzahl der Binärstellen, um die die Festpunktzahl nach rechts verschoben wird. Diese Anzahl liegt im Bereich zwischen 0 und 63_{10}. Die höherwertigen Binärstellen von D2(B2) werden ignoriert.

Der Inhalt der beiden Register R1 und R1+1 wird als 64 Bit lange Festpunktzahl mit Vorzeichen behandelt. Das Vorzeichen an der Bitstelle 0 des (geradzahligen) Registers R1 bleibt unverändert, aber alle übrigen 63 Bitstellen werden verschoben. Links freiwerdende Bitstellen werden mit dem Wert des Vorzeichenbit gefüllt, rechts über die Bitstelle 31 von R1+1 hinausgeschobene Binärstellen gehen verloren.

Anzeige

0~Zero	Verschobene Festpunktzahl = 0
1~Minus	Verschobene Festpunktzahl < 0 (Bit 0 von R1 =1)
2~Plus	Verschobene Festpunktzahl > 0 (Bit 0 von R1 =0)
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßfehler	X’5C’	R1 nicht geradzahlig.

Programmierhinweise

- Wenn B2=0 ist, bestimmt D2 allein die Verschiebeanzahl; in diesem Fall darf die Angabe von B2 im Assemblerformat entfallen.
- Wenn die Verschiebeanzahl =0 modulo 64 ist, werden R1 und R1+1 nicht verändert, aber es wird die Anzeige gesetzt.
- Die Rechtsverschiebung von negativen Festpunktzahlen führt zur "Abwärtsrundung" auf die nächstniedrige negative Ganzzahl. So ergibt z.B. die Zahl -1 bei jedweder Rechtsverschiebung wieder die Zahl -1 oder es ergibt die Zahl -5, wenn sie um 2 Binärstellen nach rechts geschoben, also durch 4 dividiert wird, die Zahl -2 (und nicht -1). Siehe dazu die Beispiele.

Beispiele

Die folgenden Beispielbefehle ergeben:

Register 0,1 vorher	Beispielbefehl	Register 0,1 nachher	Anzeige
0...0 0..0101 (+5)	SRDA 0,2	00...0 0..01 (+1)	2
1...1 1..1011 (-5)	SRDA 0,2	11...1 1..10 (-2)	1
0...0 0..0101 (+5)	SRDA 0,3	00...0 0..00 (0)	0
1...1 1..1011 (-5)	SRDA 0,3	11...1 1..11 (-1)	1
0...0 0...001 (+1)	SRDA 0,63	00...0 0..00 (0)	0
1...1 1.....1 (-1)	SRDA 0,63	11...1 1...1 (-1)	1
01..1 1.....1 ($+2^{63}-1$)	SRDA 0,63	00...0 0...0 (0)	0
10..0 0....01 ($-2^{63}+1$)	SRDA 0,63	11...1 1...1 (-1)	1

In den Beispielen ist jeweils die Rechtsverschiebung einer positiven Festpunktzahl der Rechtsverschiebung des negativen Pendants gegenübergestellt, um die vielleicht ungewohnten Resultate zu verdeutlichen.

Shift Right Double Logical

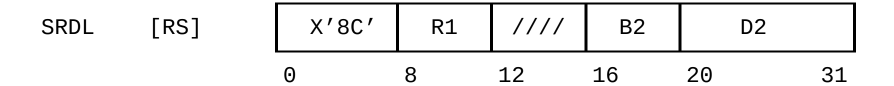
Funktion

Der Befehl SRDL verschiebt eine 64 Bit lange Binärzahl in einem Mehrzweckregister-Paar logisch um eine angegebene Anzahl von Binärstellen nach rechts.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	SRDL	R1,D2(B2)	R1 geradzahlig
* oder auch:			
	SRDL	R1,<anzahl>	R1 geradzahlig

Maschinenformat



Beschreibung

Das R1-Feld des Befehls bestimmt ein Paar von Mehrzweckregistern, bestehend aus den Registern R1 und R1+1; R1 muß geradzahlig sein, andernfalls erfolgt eine Programmunterbrechung wegen Adreßfehlers.

Die Bitstellen 12 bis 15 des Befehls werden ignoriert.

Die durch D2(B2) bestimmte Adresse wird nicht als Datenadresse verwendet; stattdessen bilden die rechten 6 Bit dieser Adresse die Anzahl der Binärstellen, um die die Binärzahl nach rechts verschoben wird. Diese Anzahl liegt im Bereich zwischen 0 und 63_{10}. Die höherwertigen Binärstellen von D2(B2) werden ignoriert.

Der Inhalt der beiden Register R1 und R1+1 wird als 64 Bit lange Binärzahl ohne Vorzeichen behandelt. Alle 64 Binärstellen dieser Zahl werden verschoben. Links freiwerdende Bitstellen werden mit 0 gefüllt. Rechts über die Bitstelle 31 von R1+1 hinausgeschobene Binärstellen gehen verloren.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßfehler	X’5C’	R1 nicht geradzahlig.

Programmierhinweise

– Wenn B2=0 ist, bestimmt D2 allein die Verschiebeanzahl; in diesem Fall darf die Angabe von B2 im Assemblerformat entfallen.

– Wenn die Verschiebeanzahl =0 modulo 64 ist, bleiben R1 und R1+1 (und die Anzeige) unverändert.

Beispiele

Die folgenden Beispielbefehle ergeben:

Register 0,1 vorher	Beispielbefehl	Register 0,1 nachher	Anzeige
0...0 0..0101	SRDL 0,2	00...0 0..0001	unverändert
1...1 1..1011	SRDL 0,2	001..1 1..1110	unverändert
1...1 1.....1	SRDL 0,63	00...0 0..0001	unverändert
01..1 1.....1	SRDL 0,63	00...0 0..0000	unverändert

Im Unterschied zum Befehl SRDA wird bei SRDL der Wert der Bitstelle 0 des Registers R1 nicht nach rechts ausgebreitet, stattdessen werden links freiwerdende Bitstellen immer mit 0 aufgefüllt.

Shift Right Single Logical

Funktion

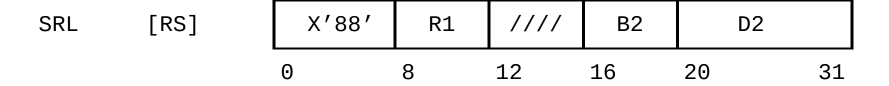
Der Befehl SRL verschiebt eine 32 Bit lange Binärzahl in einem Mehrzweckregister um eine angegebene Anzahl von Binärstellen logisch nach rechts.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	SRL	R1,D2(B2)	
* oder auch:			
	SRL	R1,<anzahl>	

Maschinenformat

SRL [RS]



Beschreibung

Der Inhalt des Mehrzweckregisters R1 wird als 32 Bit lange Binärzahl ohne Vorzeichen behandelt.

Die durch D2(B2) bestimmte Adresse wird nicht als Datenadresse verwendet; stattdessen bilden die rechten 6 Bit dieser Adresse die Anzahl der Binärstellen, um die Binärzahl nach rechts verschoben wird. Diese Anzahl liegt im Bereich zwischen 0 und 63_{10}. Die höherwertigen Binärstellen von D2(B2) werden ignoriert.

Alle 32 Binärstellen werden nach rechts verschoben. Links freiwerdende Bitstellen werden mit 0 gefüllt. Rechts über die Bitstelle 31 von R1 hinausgeschobene Binärstellen gehen verloren.

Die Bitstellen 12 bis 15 des Befehls werden ignoriert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine.

Programmierhinweise

- Wenn B2=0 ist, bestimmt D2 allein die Verschiebeanzahl; in diesem Fall darf die Angabe von B2 im Assemblerformat entfallen.
- Wenn die Verschiebeanzahl =0 modulo 64 ist, bleibt R1 (und die Anzeige) unverändert.

Beispiele

Die folgenden Beispielbefehle ergeben:

Register 0 vorher	Beispielbefehl	Register 0 nachher	Anzeige
0..0101	SRL 0,2	0....01	unverändert
1..1011	SRL 0,2	001..10	unverändert
10....0	SRL 0,31	0....01	unverändert
01....1	SRL 0,31	0.....0	unverändert

Im Unterschied zum Befehl SRA wird bei SRL der Wert der Bitstelle 0 des Registers R1 nicht nach rechts ausgebreitet, stattdessen werden links freiwerdende Bitstellen immer mit 0 aufgefüllt.

Store

Funktion

Der Befehl ST überträgt den Inhalt eines Mehrzweckregisters in ein Hauptspeicher-Wort.
Der Befehl STH überträgt die Byte 2 und 3 eines Mehrzweckregisters in ein Hauptspeicher-Halbwort.
Der Befehl STC überträgt das Byte 3 eines Mehrzweckregisters in ein Hauptspeicher-Byte.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	ST STH STC	R1,D2(X2,B2) R1,D2(X2,B2) R1,D2(X2,B2)	D2(X2,B2): Wortgrenze D2(X2,B2): Halbwortgrenze

Maschinenformate

ST	[RX]	X’50’	R1	X2	B2	D2
STH	[RX]	X’40’	R1	X2	B2	D2
STC	[RX]	X’42’	R1	X2	B2	D2
		0	8	12	16	20 31

Beschreibung

ST: Der Inhalt des Mehrzweckregisters R1 wird in das mit D2(X2,B2) adressierte Wort gespeichert.

STH: Die Bitstellen 16 bis 31 des Mehrzweckregisters R1 werden in das durch D2(X2,B2) adressierte Halbwort gespeichert.

STC: Die Bitstellen 24 bis 31 des Mehrzweckregisters R1 werden in das mit D2(X2,B2) adressierte Byte gespeichert.

Befehl	Operand1	Operand2
ST STH STC	Byte 0 bis 3 von Register R1 Byte 2 und 3 von Register R1 Byte 3 von Register R1	mit D2(X2,B2) adressiertes Wort mit D2(X2,B2) adressiertes Halbwort mit D2(X2,B2) adressiertes Byte

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Schreibzugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	STH: D2(X2,B2) keine Halbwortgrenze ST : D2(X2,B2) keine Wortgrenze

Store Clock

Funktion

Der Befehl STCK überträgt den momentanen Wert der Tagesuhr in ein Hauptspeicher-Doppelwort.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	STCK	D2(B2)	D2(B2): Doppelwortgrenze

Maschinenformat

STCK [S]

X'B205'	B2	D2
0	16	20 31

Beschreibung

Die Tagesuhr ist eine 64 Bit lange Binärzahl ohne Vorzeichen in einem internen Register der Zentraleinheit. Nach jeder Mikrosekunde, d.h. alle 10^{-6} Sekunden, wird diese Binärzahl um den Wert 4096 (2^{12}) logisch erhöht. Durch den Befehl STCK wird der momentane Wert dieser Binärzahl in das mit D2(B2) adressierte Doppelwort des Hauptspeichers übertragen.

Manche Zentraleinheiten verfügen über eine feinere Auflösung der Tagesuhr. Dies wird durch häufigere Erhöhung um einen kleineren Wert als 4096 angezeigt. In jedem Falle wird aber alle 10^{-6} Sekunden die Bitstelle 51 der Tagesuhr um Eins erhöht.

Anzeige

0~Zero Uhrzeit relativ zum 1.1.1900, 0:00 Uhr gesetzt.
1 - 3 Nicht verwendet unter BS2000.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Adreßfehler	X'48' X'5C'	Schreibzugriff auf Operand2 unmöglich D2(B2) keine Doppelwortgrenze.

Programmierhinweise

- Die Tagesuhr eignet sich zur Ermittlung des absoluten Zeitabstandes zwischen zwei Ereignissen. Zur Ermittlung des prozeßspezifischen Zeitverbrauchs dient der BS2000-Makro GEPRT. Das Tagesdatum und die Uhrzeit liefert der BS2000-Makro GDATE.
- Bei jeder System-Initialisierung des BS2000 wird die Tagesuhr auf einen Wert relativ zum Basisdatum 1. Januar 1900, 0:00 Uhr eingestellt, so daß der Binärwert 0 der Tagesuhr diesem Datum entspricht. Demzufolge enthält die Tagesuhr z.B. am 1. Januar 1987 um 0:00 den Wert $(87*365+21)*24*60*60*10^6*2^{12}$ = $(9C\ 0F\ 80\ D6\ C0\ 00\ 00\ 00)_{16}$.
- Aus der Erhöhung der Tagesuhr um 2^{12} nach jeder Mikrosekunde folgt, daß die Bitstelle 31 alle 1,048576 Sekunden um 1 erhöht wird. Wenn keine genauere Auflösung gebraucht wird, genügt das erste Wort des von STCK gespeicherten Doppelworts.

Beispiel

```
Name        Operation   Operanden

               .
TIMEBEF     DS          D
TIMEAFT     DS          D
TIMEDIFF    DC          PL8'0.00'
               .
               .
               .
            STCK        TIMEBEF
               .

   < Ablauf des zu messenden Vorgangs >

               .
            STCK        TIMEAFT
            LM          0,1,TIMEAFT           Bilden Diff TIMEAFT-TIMEBEF
            SL          1,TIMEBEF+4
            BNL         *+6                   Übertrag - Behandlung
            BCTR        0,0
            S           0,TIMEBEF
            D           0,=F'40960000'        Skalieren auf 100-stel Sek.
            CVD         1,TIMEDIFF            Quotient in Reg 1
               .
```

Das Beispiel zeigt die Benutzung des STCK-Befehls zur Zeitmessung. Vor und nach dem zu messenden Vorgang wird die Tagesuhr gelesen. Dann wird deren Differenz gebildet. Nach dem Befehl S enthalten die Register 0 und 1 die Differenz in Vielfachen von 4’’096’000.000-stel Sekunden als 64 Bit lange Festpunktzahl. (Der rechte Teil der Differenz wird logisch, der linke vorzeichengerecht gebildet). Der Befehl D liefert die Zeit abgerundet auf 100-stel Sekunden im Mehrzweckregister 1 und der Befehl CVD konvertiert dieses Ergebnis in gepackt-dezimales Format.

Store Characters under Mask

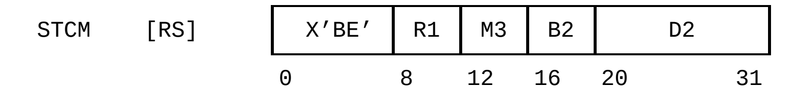
Funktion

Der Befehl STCM überträgt ausgewählte Byte eines Mehrzweckregisters in ein Feld im Hauptspeicher.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	STCM	R1,M3,D2(B2)	B'0000' ≤ M3 ≤ B'1111'

Maschinenformat



Beschreibung

Die 4 Bit der "Maske" M3 entsprechen eins zu eins den 4 Byte des Mehrzweckregisters R1 (von links nach rechts sowohl in der Maske wie im Register). Diejenigen Byte in R1, denen Einsen in der Maske gegenüberliegen, werden in aufeinanderfolgende Byte des mit D2(B2) adressierten Hauptspeicherbereichs übertragen.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	Schreibzugriff auf Operand2 unmöglich

Programmierhinweise

- Die Länge in Byte des Hauptspeicherbereichs ist gleich der Anzahl der Einsen in der Maske.
- Bei Verwendung einer Maske aus lauter Einsen (B'1111') leistet der Befehl STCM das gleiche wie der Befehl ST, aber das Hauptspeicherfeld braucht nicht an einer Wortgrenze ausgerichtet zu sein.

Beispiel

Name	Operation	Operanden
	.	
	STCM	15,B'0101',FIELD Speichern des zweiten und
*		vierten Byte von Register 15
*		in die Hauptspeicherbyte
*		FIELD und FIELD+1
	.	

Store Multiple

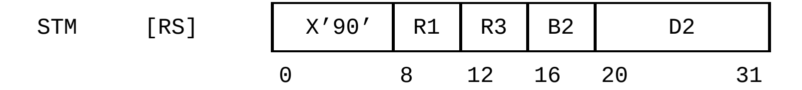
Funktion

Der Befehl STM speichert die Inhalte von (bis zu 16) aufeinanderfolgenden Mehrzweckregistern in aufeinanderfolgende Worte des Hauptspeichers.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	STM	R1,R3,D2(B2)	D2(B2): Wortgrenze

Maschinenformat



Beschreibung

Der Inhalt aufeinanderfolgender Mehrzweckregister, beginnend mit R1 und endend mit R3, wird in aufeinanderfolgende Worte des Hauptspeichers übertragen. Das erste Wort ist mit D2(B2) adressiert.
Wenn R1 > R3 ist, wird ab dem Mehrzweckregister R1 bis zum Mehrzweckregister 15 und dann ab dem Mehrzweckregister 0 bis zum Register R3 gespeichert. Wenn R1=R3 ist, wird nur ein Mehrzweckregister (R1) gespeichert.

Befehl	Operand1	Operand2
STM	Inhalte der Register R1 bis R3	mit D2(B2) adressierte Wortfolge Wortanzahl =R3-R1+1, wenn R3≥R1 =R3-R1+17, wenn R3<R1

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	Schreibzugriff auf Operand2 unmöglich
Adreßfehler	X'5C'	D2(B2) keine Wortgrenze

Beispiel

Name	Operation	Operanden
	.	
	STM	15,0,SAVE Der Inhalt der Mehrzweckregister
*		15 und 0 wird in die zwei
*		aufeinanderfolgenden Worte
*		SAVE und SAVE+4 gespeichert.
	.	

Supervisor Call

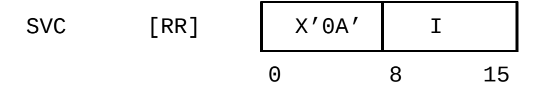
Funktion

Der Befehl SVC bewirkt den Aufruf einer (privilegierten) Routine des Betriebssystems. Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	SVC	I	X'00' ≤ I ≤ X'FF'

Maschinenformat



Beschreibung

Der Befehl SVC bewirkt einen Supervisor- ("Organisationsprogramm"-) Aufruf und überträgt dazu den Inhalt seines I-Felds in ein privilegiertes Register der Zentraleinheit. Der weitere Ablauf des Befehls findet im privilegierten Zustand der Zentraleinheiten statt und wird deshalb hier nicht weiter beschrieben.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine.

Programmierhinweise

Der Befehl SVC ist Teil der Expansion vieler BS2000-Makros, z.B. des Makros WROUT. Er besorgt den Transfer dieser Makros an die BS2000-Systemroutinen. Die Einschalung des SVC in Makros befreit den Anwender u.a. von der Memorierung des entsprechenden SVC-Codes und ermöglicht deren Portabilität z.B. beim Übergang von BS2000-Versionen.

Test under Mask

Funktion

Der Befehl TM testet ausgewählte Bitstellen eines Byte im Hauptspeicher.
Die Anzeige wird gemäß dem Testergebnis gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	TM	D1(B1),I2	B’00000000’ ≤ I2 ≤ B’11111111’

Maschinenformat

TM [SI]

X’91’	I2	B1	D1
0	8	16	20 31

Beschreibung

Der Direktoperand I2 des Befehls wird als Maske verwendet, mit der ausgewählte Bit des mit D1(B1) adressierten Byte im Hauptspeicher getestet werden. Die 8 Bit der Maske entsprechen Eins zu Eins den 8 Bit des zu testenden Byte (von links nach rechts in beiden Operanden). Jeder Bitwert 1 in der Maske wählt das entsprechende Bit im Hauptspeicherbyte aus und testet es. Wenn alle getesteten Bit =0 sind, wird die Anzeige auf 0~Zero gesetzt, wenn sie alle =1 sind, wird die Anzeige auf 3~Ones gesetzt und wenn unter den getesteten Bit sowohl solche mit dem Wert 0 als auch solche mit dem Wert 1 vorkommen, wird die Anzeige auf 1~Mixed gesetzt.

Anzeige

0~Zeroes	Alle getesteten Bit sind =0.
1~Mixed	Die getesteten Bit sind weder alle =0 noch alle =1.
2	Nicht verwendet.
3~Ones	Alle getesteten Bit sind =1.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lesezugriff auf Operand1 unmöglich

Programmierhinweise

Der Assembler [1] verfügt über "erweiterte mnemotechnische Operationscodes" für Sprungbefehle nach dem Befehl TM:

- BZ oder BRZ (Branch when Zeroes) zur Abfrage auf reines 0_2-Bitmuster
- BO oder BRO (Branch when Ones) zur Abfrage auf reines 1_2-Bitmuster
- BM oder BRM (Branch when Mixed) zur Abfrage auf gemischtes 0_2-1_2-Bitmuster

Eine vollständige Liste aller erweiterten mnemotechnischen Operationscodes findet sich im Anhang.

Beispiele

Die folgenden Beispielbefehle setzen folgende Anzeigen:

Name	Operation	Operanden	Anzeige	Abfrage z.B. durch
	.			
TESTBYTE	DC	X'87'		
	.			
	.			
	.			
Beispiel1	TM	TESTBYTE,X'87'	3	BO, BRO
Beispiel2	TM	TESTBYTE,X'70'	0	BZ, BRZ
Beispiel3	TM	TESTBYTE,X'88'	1	BM, BRM
	.			

Translate

Funktion

Der Befehl TR setzt die Byte eines Zielfeldes gemäß einer Umsetzungstabelle um.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	TR	D1(L,B1),D2(B2)	1 ≤ L ≤ 256

Maschinenformat

TR [SS]

X'DC'	L-1	B1	D1	B2	D2
0	8	16	20	32	36 47

Beschreibung

Mit D1(B1) wird das "Zielfeld" (der Länge L Byte) adressiert, mit D2(B2) die "Umsetzungstabelle". Das Zielfeld enthält vor der Befehlsausführung die umzusetzenden und nach der Befehlsausführung die umgesetzten Byte.

Das Zielfeld wird von links nach rechts byteweise verarbeitet. Jedes Byte des Zielfelds ("Argumentbyte") wird einzeln zur Anfangsadresse der Umsetzungstabelle addiert. Die Summe adressiert ein Byte ("Funktionsbyte") der Umsetzungstabelle. Das Funktionsbyte ersetzt dann das ursprüngliche Argumentbyte im Zielfeld. Der Befehl wird beendet, wenn alle Argumentbyte durch Funktionsbyte ersetzt sind. Die Addition jedes Argumentbytes zur Anfangsadresse der Umsetzungstabelle erfolgt logisch, wobei das Argumentbyte als vorzeichenlose 8 Bit Zahl aufgefaßt wird; die Summe ist - je nach Adressierungsmodus - entweder 24 Bit oder 31 Bit lang.

Die Umsetzungstabelle wird nicht verändert, es sei denn, sie überlappt sich mit dem Zielfeld. Bei Überlappung erfolgt keine Anzeige, aber i.a. sind frühere Byte-Umsetzungen durch spätere wieder verändert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.

Programmierhinweise

- Der Befehl TR kann dazu benutzt werden, um Eingabedaten von einem Code in einen anderen Code zu konvertieren, z.B. von ASCII in EBCDIC oder von Kleinbuchstaben in Großbuchstaben (siehe Beispiel 1).
- Der Befehl TR kann auch dazu benutzt werden, um die Byte eines Zielfeldes in eine andere Reihenfolge zu bringen. Dazu speichert man in das Zielfeld ein "Muster" und verwendet als "Umsetzungstabelle" das Sendefeld mit den in ihrer Reihenfolge zu ändernden Byte. Das Byte *i* des Musters (*i*=0,1,...,L-1) muß dabei die (binäre) Platznummer desjenigen Byte im Sendefeld enthalten, das an das Byte *i* des Zielfelds kommen soll. Wenn beispielsweise ein 3 Byte langes Zielfeld "zyklisch vertauscht" werden soll, also z.B. aus a, b, c die Folge c, a, b gemacht werden soll, muß das Byte 0 des Zielfelds, d.h. des Musters, den Wert X’02’ (=Platznummer des Sendebytes ’c’) erhalten, Byte 1 den Wert X’00’ und Byte 2 den Wert X’01’ (siehe dazu das Beispiel 2).
- Die Umsetzungstabelle ist so lang wie der Wert des größten Argumentbyte plus 1. Sicherheitshalber wird man gewöhnlich die Maximallänge von 256 Byte für jede Umsetzungstabelle vorsehen und auch alle Byte in ihr mit Werten definieren. Nur wenn sicher bekannt ist, daß der Wert des größten Argumentbyte kleiner als $(FF)_{16}$ sein wird oder daß einzelne Argumentbyte-Werte nicht vorkommen werden, kann man von diesem Brauch abweichen.

Beispiele

Beispiel 1

Um eine Zeichenfolge CHARFLD aus Kleinbuchstaben in Großbuchstaben zu übersetzen (EBCDI-Code unterstellt), kann folgender TR-Befehl verwendet werden:

Name	Operation	Operanden
	.	
CONVTB	DS	0C
	ORG	CONVTB+'a'
	DC	'ABCDEFGHI'
	ORG	CONVTB+'j'
	DC	'JKLMNOPQR'
	ORG	CONVTB+'s'
	DC	'STUVWXYZ'
	ORG	
	.	
	.	
	.	
	TR	CHARFLD,CONVTB
	.	

Beispiel 2

Die Übertragung eines Zeichenfelds ORGFIELD in ein Zeichenfeld INVFIELD unter "Invertierung" seines Inhalts, d.h. so, daß das letzte Byte von ORGFIELD zum ersten Byte in INVFIELD, das vorletzte Byte von ORGFIELD zum zweiten Byte von INVFIELD wird, u.s.f., kann so erfolgen:

Name	Operation	Operanden	
	.		
	LA	0,L'ORGFIELD	Füllen INVFIELD
	LA	1,0	mit Platznummern
MUSTERN	BCTR	0,0	L-1, L-2,..., 0
	STC	0,INVFIELD(1)	
	LA	1,1(1)	
	LTR	0,0	
	BNE	MUSTERN	
	TR	INVFIELD(L'ORGFIELD),ORGFIELD	
	.		

Diese Befehle erzeugen z.B. aus

ORGFIELD	DC	'0123456789'

das invertierte Feld

INVFIELD	DC	'9876543210'

Translate and Test

Funktion

Der Befehl TRT testet die Byte eines Zielfeldes gemäß einer Umsetzungstabelle.
Die Anzeige wird gemäß dem Testergebnis gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	TRT	D1(L,B1),D2(B2)	1 ≤ L ≤ 256

Maschinenformat

TRT [SS]

X'DD'	L-1	B1	D1	B2	D2
0	8	16	20	32	36 47

Beschreibung

Mit D1(B1) wird das "Zielfeld" der Länge L Byte adressiert, mit D2(B2) die "Umsetzungstabelle".

Das Zielfeld wird von links nach rechts byteweise verarbeitet. Jedes Byte des Zielfelds ("Argumentbyte") wird einzeln zur Anfangsadresse der Umsetzungstabelle addiert. Die Summe adressiert ein Byte ("Funktionsbyte") der Umsetzungstabelle. Wenn dieses Funktionsbyte $\neq 00_{16}$ ist, wird der Befehl beendet, andernfalls wird das nächste Argumentbyte bearbeitet.

Wenn alle Funktionsbyte der Umsetzungstabelle $=00_{16}$ sind, wird der Befehl mit der Anzeige 0~Zero beendet.

Das erste Funktionsbyte, das $\neq 00_{16}$ ist, beendet den Befehl und zwar mit der Anzeige 2~Plus, wenn das zugehörige Argumentbyte das letzte Byte des Zielfelds war, andernfalls mit der Anzeige 1~Minus. Die Adresse des zugehörigen Argumentbyte wird in das Mehrzweckregister 1 und das von Null verschiedene Funktionsbyte wird in das Mehrzweckregister 2 eingetragen.

Die Addition jedes Argumentbyte zur Anfangsadresse der Umsetzungstabelle erfolgt logisch, wobei das Argumentbyte als vorzeichenlose 8-Bit-Zahl aufgefaßt wird; die Summe ist - je nach Adressierungsmodus - entweder 24 Bit oder 31 Bit lang.

Weder die Umsetzungstabelle noch das Zielfeld werden verändert, auch nicht bei Überlappung (die zulässig ist).

Die Mehrzweckregister 1 und 2 werden nur verändert, wenn ein Funktionsbyte $\neq 00_{16}$ auftritt.
Im 24-Bit-Adressierungsmodus wird die 24 Bit lange Argumentbyte-Adresse in die Bitstellen 8 bis 31 des Mehrzweckregisters 1 eingetragen und die Bitstellen 0 bis 7 werden nicht verändert, im 31-Bit-Adressierungsmodus wird die 31 Bit lange Argumentbyte-Adresse in die Bitstellen 1 bis 31 des Registers 1 eingetragen und die Bitstelle 0 wird auf 0 gesetzt.
Der Wert des ersten von Null verschiedenen Funktionsbyte wird in die Bitstellen 24 bis 31 des Mehrzweckregisters 2 eingetragen; die Bitstellen 0 bis 23 bleiben unverändert.

Anzeige

0~Zero	Alle Funktionsbyte sind $=00_{16}$.
1~Minus	Ein Funktionsbyte ungleich 00_{16} wurde entdeckt, bevor das letzte Argumentbyte des Zielfelds verarbeitet wurde.
2~Plus	Das Argumentbyte, das zum letzten Byte des Zielfelds gehört, war $\neq 00_{16}$.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lesezugriff auf Operand1 oder Operand2 unmöglich

Programmierhinweise

- Die Umsetzungstabelle ist so lang wie der Wert des größten verarbeiteten Argumentbyte plus 1.
- Im Gegensatz zum Befehl TR bleibt beim Befehl TRT die Zieltabelle unverändert.
- Der Befehl TRT benutzt die Mehrzweckregister 1 und 2, obwohl diese nicht im Befehl angegeben werden.
- Weil die Mehrzweckregister 1 und 2 nicht in jedem Falle geändert werden und auch dann, wenn sie geändert werden, nicht vollständig ersetzt werden, empfiehlt es sich, diese Register 1 und 2 *vor* TRT explizit zu besetzen, z.B. auf die Adresse des ersten Byte nach dem zu bearbeitenden Speicherbereich.
- Der Befehl TRT kann dazu benutzt werden, um ein Zielfeld auf Zeichen hin zu überprüfen, die eine besondere Bedeutung haben, z.B. unzulässig sind. Dazu setzt man in der Umsetzungstabelle alle Funktionsbyte, die solchen Zeichen entsprechen, auf einem Wert $\neq 00_{16}$ und alle übrigen Funktionsbyte auf den Wert $=00_{16}$.

Beispiel

Das Zielfeld DFIELD soll untersucht werden, ob in ihm die Zeichen "+" oder "-" vorkommen. Dies kann geschehen durch die Datenerklärung

Name	Operation	Operanden	
	.		
CONVTB	DC	256X’00’	Funktionsbyte 00 für übrige
	ORG	CONVTB+’+’	Zeichen
	DC	X’01’	Funktionsbyte 01 für +
	ORG	CONVTB+’-’	
	DC	X’02’	Funktionsbyte 02 für -
	ORG		
	.		

zusammen mit den Befehlen:

	.		
	SR	1,1	Löschen Register 1 und 2
	SR	2,2	siehe Programmierhinweise
	TRT	DFIELD,CONVTB	
	BE	NOSIGN	Kein + oder - in DFIELD
	BH	TRAILING	+ oder - im letzten Byte
	BL	EMBEDDED	+ oder -, aber nicht im letzten Byte
	.		

Im Falle von TRAILING und EMBEDDED enthält das Mehrzweckregister 1 die (je nach Adressierungsmodus entweder 24 Bit lange oder 31 Bit lange) Adresse des ersten + oder - in DFIELD und das Mehrzweckregister 2 in seinem niedrigstwertigen Byte das zugehörige Funktionsbyte, hier also entweder 01_{16} oder 02_{16}. Man beachte die abschließende ORG-Anweisung: sie verhindert, daß eine etwa folgende Datenerklärung in CONVTB hineinreicht.

Test and Set

Funktion

Der Befehl TS setzt die Anzeige gemäß dem Wert des höchstwertigen Bit eines Hauptspeicher-Byte und überschreibt dann dieses Byte mit $(FF)_{16}$. Während seines Ablaufs ist der Befehl nicht unterbrechbar.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	TS	D2(B2)	

Maschinenformat

TS [S]

X'93'	///////	B2	D2
0	8	16	20 31

Beschreibung

Das höchstwertige Bit des durch D2(B2) adressierten Hauptspeicher-Byte wird getestet. Wenn es =0 ist, wird die Anzeige auf 0~Zero gesetzt, andernfalls auf 1~Not Zero. Anschließend werden alle Bit des Byte auf 1 gesetzt, also das Byte mit FF_{16} überschrieben. Die Bitstellen 8 bis 15 des Befehls werden ignoriert.

Die Besonderheit des Befehls TS besteht darin, daß im Zeitraum zwischen dem Test des höchstwertigen Bit des adressierten Byte und dem Abschluß des Überschreibens mit $(FF)_{16}$ keine andere Zentraleinheit und kein Kanal auf das Byte Schreib- oder Lesezugriff hat. Zu diesem Zweck findet vor und nach dem Befehl in der Hardware eine sog. Serialisierung statt, bei der alle anstehenden Speicherzugriffe abgearbeitet werden. Dieser Mechanismus prädestiniert den Befehl TS für Synchronisationsprobleme in Multiprozessor-Anwendungen.

Anzeige

0~Zero	Höchstwertiges Bit von D2(B2) war =0.
1~Not Zero	Höchstwertiges Bit von D2(B2) war =1.
2	Nicht verwendet.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lese-Schreibzugriff auf Operand2 unmöglich

Programmierhinweise

Der Befehl TS ist weniger mächtig als die Befehle CS und CDS und nur aus Kompatibilätsgründen im Befehlsvorrat enthalten. Deswegen wird für weitere Erläuterungen auf die Beschreibung der Befehle CS und CDS sowie auf den Anhang 7.6 verwiesen.

Unpack

Funktion

Der Befehl UNPK erzeugt aus einer (gepackten) Dezimalzahl des Sendefelds eine entpackte (gezonte) Dezimalzahl im Empfangsfeld.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	UNPK	D1(L1,B1),D2(L2,B2)	1 ≤ L1,L2 ≤ 16

Maschinenformat

UNPK [SS]

X'F3'	L1-1	L2-1	B1	D1	B2	D2
0	8	12	16	20	32	36 47

Beschreibung

Durch D1(L1,B1) ist das Empfangsfeld, durch D2(L2,B2) ist das Sendefeld adressiert (1 ≤ L1,L2 ≤ 16). Die im Sendefeld enthaltene (gepackte) Dezimalzahl wird ins Empfangsfeld übertragen und dabei ins entpackte (gezonte) Format umgewandelt.

Das Sendefeld wird *nicht* darauf geprüft, ob es wirklich eine korrekte, gepackte Dezimalzahl enthält, sondern wird so behandelt, als enthalte sie eine.

Beide Operanden werden byteweise von rechts nach links verarbeitet. Zunächst werden die beiden Halbbyte des niedrigstwertigen Byte des Sendefelds in ihrer Reihenfolge vertauscht in das niedrigstwertige Byte des Empfangsfelds übertragen. Danach wird jedes weitere Halbbyte des Sendefelds in das rechte Halbbyte eines Byte des Empfangsfelds übertragen und das jeweils linke Halbbyte ("Zone") = F_{16} gesetzt.

Wenn das Sendefeld vor dem Empfangsfeld ausgeschöpft ist, d. h., wenn 2L2 < L1+1 ist, werden die linken (L1-2L2+1) Byte des Empfangsfelds mit $(F0)_{16}$ gefüllt; wenn das Empfangsfeld zu kurz ist zur Aufnahme aller Halbbyte des Sendefelds, d.h., wenn L1 < 2L2-1 ist, werden die linken (2L2-L1-1) Halbbyte des Sendefelds ignoriert.

Empfangsfeld und Sendefeld dürfen sich überlappen. Dabei verändern im allgemeinen spätere Byte-Operationen das Ergebnis früherer Byte-Operationen desselben Befehls. Der Befehl wird so ausgeführt, als werde jedes Byte des Empfangsfelds unmittelbar dann gespeichert, wenn das dafür benötigte Byte des Sendefelds gelesen worden ist.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.

Programmierhinweise

Das Sendefeld wird nur geändert, wenn es sich mit dem Empfangsfeld überlappt.

Beispiele

Die folgenden Beispiele von UNPK-Befehlen liefern folgende Ergebnisse:

FIELD vorher	Beispiel-Befehl	FIELD nachher	Anzeige
ohne Belang	UNPK FIELD(1),=PL1’-3’	Z’-3’	unverändert
ohne Belang	UNPK FIELD(5),=PL2’12’	Z’00012’	unverändert
X’89’	UNPK FIELD(1),FIELD(1)	X’98’	unverändert
X’23456789’	UNPK FIELD(4),FIELD(4)	X’F6F6F798’	unverändert

Das dritte Beispiel nützt aus, daß das Sendefeld (FIELD) nicht auf gepacktes Format geprüft wird (X’89’ ist keine korrekte gepackte Dezimalzahl): es werden einfach die beiden Halbbyte von FIELD vertauscht. Eine solche Vertauschung findet allerdings nur im letzten (hier: einzigen) Byte statt, wie man am vierten Beispiel sieht. Bei diesem vierten Beispiel ist übrigens einer der Fälle von sich überlappenden Operanden gezeigt, bei denen eine spätere Byte-Operation (hier die dritte) das Ergebnis einer früheren Byte-Operation (hier der zweiten) ändert. (Der Sinn des Beispiels ist, diesen Fall zu zeigen, nicht, ihn zu empfehlen.)

Exclusive Or

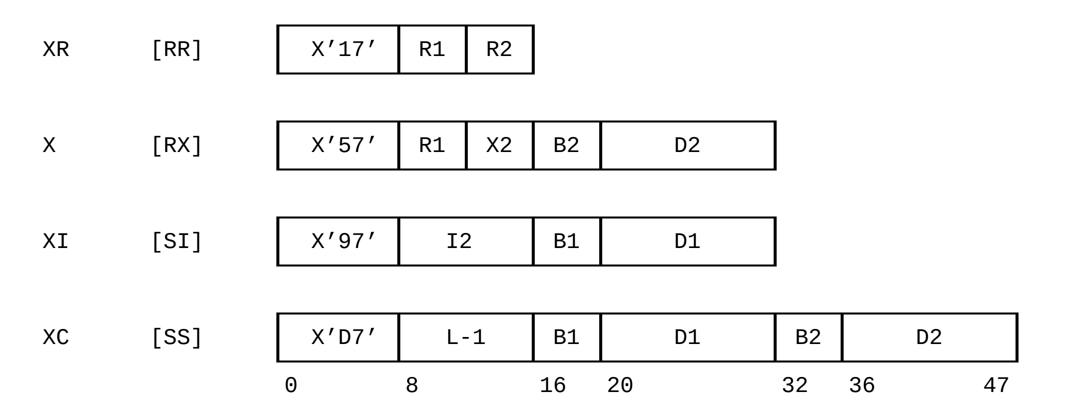
Funktion

Die Befehle XR, X, XI und XC bewirken bitweise die logische EXKLUSIV ODER-Verknüpfung zweier Operanden.
Die Anzeige wird gemäß dem Wert des Ergebnisses gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	XR X XI XC	R1,R2 R1,D2(X2,B2) D1(B1),I2 D1(L,B1),D2(B2)	 D2(X2,B2): Wortgrenze X’00’ ≤ I2 ≤ X’FF’ 1 ≤ L ≤ 256

Maschinenformate



Beschreibung

Die Bit des ersten Operanden werden gemäß der folgenden Tabelle durch die gegenüberliegenden Bit des zweiten Operanden verändert. Das Ergebnis ersetzt den ersten Operanden.

Tabelle der EXKLUSIV ODER-Verknüpfungen

Bitwert im ersten Operanden	Bitwert im zweiten Operanden	Bitwert im Ergebnis
0	0	0
0	1	1
1	0	1
1	1	0

Operanden

Befehl	Operand1	Operand2
XR	Inhalt von Register R1	Inhalt von Register R2
X	Inhalt von Register R1	mit D2(X2,B2) adressiertes Wort
XI	mit D1(B1) adressiertes Byte	Direktoperand I2
XC	mit D1(B1) adressiertes Feld der Länge L Byte	mit D2(B2) adressiertes Feld der Länge L Byte

Anzeige

0~Zero	Ergebnis =0
1~Not Zero	Ergebnis ≠0
2	Nicht verwendet.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	X: Lesezugriff auf Operand2 unmöglich XI: Lese-Schreibzugriff auf Operand1 unmöglich XC: Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	X: D2(X2,B2) keine Wortgrenze

Programmierhinweise

- Die EXCLUSIV OR-Befehle invertieren im ersten Operanden alle Bitstellen, denen im zweiten Operanden eine Bitstelle mit dem Wert 1 gegenüberliegt und lassen die übrigen Bitstellen des ersten Operanden unverändert.
- Die Verarbeitung der Operanden erfolgt byteweise von links nach rechts.
- Bei XC dürfen die Operanden sich überlappen. Dabei werden allerdings i.a. frühere Byte-Operationen durch spätere wieder geändert.
- Wenn beim XR-Befehl R1=R2 ist, werden das Mehrzweckregister R1 und die Anzeige auf Null gesetzt.
- Der Befehl XC mit Operand1=Operand2 setzt alle Byte von Operand1 auf X’00’.
- Bei Anwendungen der Befehle XI und XC in Multiprozessor-Anlagen ist folgendes zu beachten:
  Speicherzugriffe des ersten Operanden der Befehle XI und XC bestehen aus dem Lesen eines Byte aus dem Speicher und dem anschließenden Abspeichern des veränderten Wertes in den Speicher. Diese Lese- und Abspeicherzugriffe auf ein einzelnes Byte geschehen nicht unbedingt sofort hintereinander, wenn ein weiterer Prozessor oder ein weiteres Programm (oder ein Kanalprogramm, Ein-/Ausgabe) versucht, diese Speicherstelle zu ändern. Eine sichere Methode zum Update eines gemeinsam benutzten Speicherworts ist im Anhang 7.6 sowie in den Programmierhinweisen der Befehle CS und CDS beschrieben.

Beispiel

Name	Operation	Operanden
	.	
	XC	A,B Dieser berühmte Trick tauscht
	XC	B,A die Hauptspeicherfelder
	XC	A,B A und B ohne Hilfsfeld aus.
	.	

Der schöne Algorithmus enthält übrigens eine Ausnahme: wenn sich die beiden Bereiche A und B überlappen (oder identisch sind), wird der gemeinsame Teil mit binären Nullen gefüllt anstatt unverändert zu bleiben.
Man kann auf die gleiche Weise (mit 3 Befehlen XR) die Inhalte zweier (verschiedener) Mehrzweckregister austauschen, allerdings nicht ein Hauptspeicherwort mit einem Mehrzweckregister, weil es keinen X-Befehl gibt, dessen *erster* Operand ein Hauptspeicherwort adressiert.

4 Dezimalbefehle

Überblick

Die Dezimalbefehle dienen

a) der Addition, Subtraktion, Multiplikation, Division und dem Vergleich von zwei Dezimalzahlen (AP, SP, MP, DP, CP),

b) der Verschiebung und/oder Rundung einer Dezimalzahl (SRP),

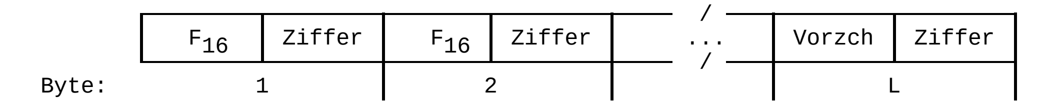
c) der Druckaufbereitung einer oder mehrerer aufeinanderfolgender Dezimalzahlen (ED, EDMK).

Die von den Dezimalbefehlen verarbeiteten Dezimalzahlen sind vorzeichenbehaftete Ganzzahlen zur Basis 10 mit maximal 31 Stellen. Sie werden bei allen Dezimalbefehlen dem Hauptspeicher entnommen bzw. dorthin erzeugt. Es gibt keine Register-Operanden, allerdings sind bei einigen Dezimalbefehlen Mehrzweckregister für besondere Anzeigen beteiligt.

Die Länge einer Dezimalzahl (in Byte) wird bei allen Dezimalbefehlen im Befehl selbst angegeben und ist nicht Teil der Dezimalzahl selbst. (Der Assembler [1] erleichtert die Längenbestimmung erheblich).

Die von den Dezimalbefehlen verarbeiteten Dezimalzahlen sind in einem von zwei Formaten im Hauptspeicher gespeichert, entweder im entpackten oder im gepackten Format. Das entpackte Format (in anderem Zusammenhang auch "gezontes" Format genannt) ist das Format nach der Eingabe (z.B. von einer Tastatur) oder für die Ausgabe (z.B. auf einen Drucker), das gepackte Format ist das (vorausgesetzte) Format für die arithmetische Behandlung oder für den Vergleich von Dezimalzahlen. (Im Assembler sind Dezimalzahlen in beiden Formaten definierbar und zwar durch die Konstantentypen Z bzw. P (siehe unten)). Nachfolgend werden die beiden Formate näher erläutert.

Entpacktes Format



Beim entpackten (oder "gezonten") Format von Dezimalzahlen ist jede Dezimalstelle in einem Byte dargestellt: die Einerstelle im letzten, L-ten Byte, die Zehnerstelle im vorletzten, (L-1)-ten Byte, die Hunderterstelle im vorvorletzten, (L-2)-ten Byte u.s.f..

Der numerische Wert jeder Dezimalstelle ist durch sein sedezimales Äquivalent repräsentiert ($d_{10} = d_{16}$, d =0, 1, ... 9) und bildet jeweils das rechte Halbbyte. Das linke Halbbyte (die "Zone") der ersten L-1 Byte ist konstant = F_{16}, das linke Halbbyte des letzten, L-ten Byte enthält das Vorzeichen.

Ein positives Vorzeichen wird durch die Sedezimalwerte A_{16} oder C_{16} oder E_{16} oder F_{16} bestimmt, ein negatives Vorzeichen durch die Sedezimalwerte B_{16} oder D_{16}.

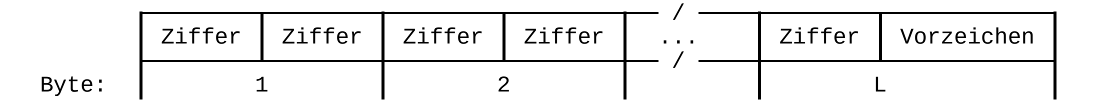
Ein L Byte langes entpacktes Format kann eine bis zu L-stellige Dezimalzahl aufnehmen, wobei die linken Byte den Wert $(F0)_{16}$ enthalten, wenn die Dezimalzahl weniger als L Dezimalstellen umfaßt. Die maximale Länge einer Dezimalzahl des entpackten Formats beträgt 16 Byte, so daß bis zu 16-stellige Dezimalzahlen in diesem Format dargestellt werden können.

Beispiele:

Dezimalzahl	entpacktes Format	
+12	F1 A2 oder F1 C2 oder F1 E2 oder F1 F2	(2 Byte)
-5	B5 oder D5	(1 Byte)

Im Assembler [1] wird das entpackte Format einer Dezimalzahl durch den Konstantentyp Z erzeugt, wobei zur Vorzeichendarstellung die Sedezimalwerte C_{16} (für plus) und D_{16} (für minus) eingesetzt werden: Z'12' oder Z'+12' definieren beide je 2 Byte mit dem Inhalt $(F1\ C2)_{16}$ und Z'-5' definiert 1 Byte mit dem Inhalt $(D5)_{16}$. Man kann die Bestimmung der Länge einer Dezimalzahl dem Assembler überlassen (wie in den gerade gezeigten Beispielen) oder eine explizite Länge angeben: ZL3'12' definiert 3 Byte mit dem Inhalt $(F0\ F1\ C2)_{16}$. Der Programmierer kann auch bei der Definition einen Dezimalpunkt einfügen, der aber weder in der Speicherdarstellung noch bei den Dezimalbefehlen berücksichtigt wird.

Gepacktes Format



Beim gepackten Format von Dezimalzahlen ist jede Dezimalstelle in einem Halbbyte dargestellt. Die Einerstelle ist im linken Halbbyte des letzten, L-ten Byte, die Zehnerstelle im rechten Halbbyte des vorletzten, (L-1)-ten Byte, die Hunderterstelle im gleichen, (L-1)-ten Byte, aber in dessen linken Halbbyte gespeichert, u.s.f.

Der numerische Wert jeder Dezimalstelle ist durch sein sedezimales Äquvalent repräsentiert ($d_{10} = d_{16}$, d =0, 1, ... 9). Das Vorzeichen der Dezimalzahl ist im rechten Halbbyte des letzten (niedrigstwertigen) Byte dargestellt. Ein positives Vorzeichen wird durch die Sedezimalwerte A_{16} oder C_{16} oder E_{16} oder F_{16} bestimmt, ein negatives Vorzeichen durch die Sedezimalwerte B_{16} oder D_{16}.

Ein L Byte langes gepacktes Format kann eine bis zu (2L-1)-stellige Dezimalzahl aufnehmen, wobei die oberen Halbbyte $=0_{16}$ enthalten, wenn die Dezimalzahl weniger als (2L-1) signifikante Dezimalstellen umfasst. Die maximale Länge des gepackten Formats einer Dezimalzahl beträgt 16 Byte, so daß bis zu 31-stellige Dezimalzahlen in diesem Format dargestellt werden können. Der absolute Wertebereich W von gepackten Dezimalzahlen beträgt demnach: $0 \leq W \leq 10^{31}-1$

Beispiele:

Dezimalzahl	gepacktes Format	
+12	01 2A oder 01 2C oder 01 2E oder 01 2F	(2 Byte)
-5	5B oder 5D	(1 Byte)

Im Assembler wird das gepackte Format einer Dezimalzahl durch den Konstantentyp P erzeugt, wobei zur Vorzeichendarstellung die Sedezimalwerte C_{16} (für plus) und D_{16} (für minus) eingesetzt werden: P’12’ oder P’+12’ definieren beide je 2 Byte mit dem Inhalt $(01\ 2C)_{16}$ und P’-5’ definiert 1 Byte mit dem Inhalt $(5D)_{16}$. Wie beim entpackten Format bestimmt der Assembler implizit die Länge, wenn eine explizite Längenangabe fehlt.

Tabelle der Vorzeichenverschlüsselungen

Verschlüsselung	Vorzeichen
A_{16}	positiv
B_{16}	negativ
C_{16}	positiv
D_{16}	negativ
E_{16}	positiv
F_{16}	positiv

Add Decimal

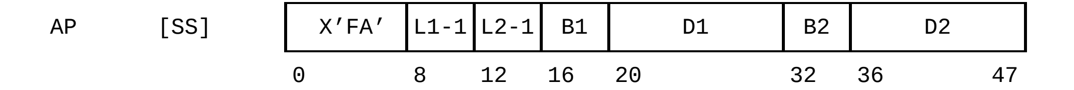
Funktion

Der Befehl AP addiert zwei gepackte Dezimalzahlen. Die Summe ersetzt den ersten Summanden.
Die Anzeige wird gemäß dem Wert der Summe gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	AP	D1(L1,B1),D2(L2,B2)	1 ≤ L1,L2 ≤ 16

Maschinenformat



Beschreibung

Die gepackte Dezimalzahl im Feld des zweiten Operanden (D2(L2,B2)) wird zur gepackten Dezimalzahl im Feld des ersten Operanden (D1(L1,B1)) vorzeichengerecht addiert.
Die gepackte Summe ersetzt den ersten Operanden. Der erste Operand und die Summe haben die Länge L1 Byte, der zweite Operand hat die Länge L2 Byte, 1 ≤ L1, L2 ≤ 16.
Beide Operanden werden auf korrektes gepacktes Format geprüft, im Fehlerfalle erfolgt eine Programmunterbrechung wegen Datenfehlers.
Ein Dezimal-Überlauf tritt auf, wenn die Summe mehr signifikante Dezimalstellen aufweist als in das Feld des ersten Operanden passen. In diesem Fall wird der Befehl normal beendet, es werden jedoch nur die 2L1-1 niedrigstwertigen Dezimalstellen der Summe gespeichert und die höchstwertigen Dezimalstellen gehen verloren. Die Anzeige wird auf 3~Overflow gesetzt. Wenn das Bit für Dezimal-Überlauf in der Programmaske auf 1 gesetzt ist (BS2000-Standard), erfolgt außerdem danach eine Programmunterbrechung wegen Dezimal-Überlaufs.

Eine echte Summe =0 hat stets ein positives Vorzeichen (C_{16}), jedoch kann eine 0, die durch Dezimal-Überlauf entstanden ist, auch ein negatives Vorzeichen (D_{16}) haben.

Anzeige

0~Zero Summe = 0 (mit dem Vorzeichen C_{16}).
1~Minus Summe < 0 (mit dem Vorzeichen D_{16}).
2~Plus Summe > 0 (sie hat das Vorzeichen C_{16}).
3~Overflow Summe zu groß.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.
Datenfehler	X’60’	Inkorrektes Format eines Summanden
Dezimal-Überlauf	X’74’	Summe zu groß für den ersten Operanden

Programmierhinweise

- Beide Operanden werden als Ganzzahlen verarbeitet.
- Ein positives Vorzeichen des Resultats wird durch C_{16}, ein negatives Vorzeichen durch D_{16} dargestellt.
- Die beiden Operanden dürfen sich überlappen, aber dann müssen die Adressen ihrer niedrigstwertigen Byte gleich sein (D1(B1)+L1-1 = D2(B2)+L2-1), andernfalls erfolgt eine Programmunterbrechung wegen Datenfehlers.
- Der zweite Operand wird nur geändert, wenn er sich mit dem ersten Operanden überlappt.
- Die Verarbeitung der Operanden geschieht von rechts nach links.
- Bei einem Dezimal-Überlauf hat das Resultat das Vorzeichen der korrekten Summe.

Beispiele

Die folgenden Beispiele von AP-Befehlen ergeben folgende Ergebnisse:

DFIELD vorher	Beispiel-Befehl	DFIELD nachher	Anzeige
PL1'+1'	AP DFIELD,=PL1'-1'	PL1'+0'	0
PL1'+1'	AP DFIELD,=PL16'-2'	PL1'-1'	1
PL1'+1'	AP DFIELD,DFIELD	PL1'+2'	2
PL1'+1'	AP DFIELD,=PL8'-11'	PL1'-0'	3

Man beachte, daß der Dezimal-Überlauf im vierten Beispiel nicht durch die Überlänge des zweiten Operanden ausgelöst wird, sondern deshalb auftritt, weil die Summe (-10) nicht in den ersten Operanden paßt. Bei Dezimal-Überlauf kann es - wie hier - geschehen, daß eine resultierende Null ein negatives Vorzeichen bekommt.

Compare Decimal

Funktion

Der Befehl CP vergleicht zwei gepackte Dezimalzahlen.
Die Anzeige wird gemäß dem Vergleichsergebnis gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	CP	D1(L1,B1),D2(L2,B2)	1 ≤ L1,L2 ≤ 16

Maschinenformat

CP [SS]

X'F9'	L1-1	L2-1	B1	D1	B2	D2
0	8	12	16	20	32	36 47

Beschreibung

Die gepackte Dezimalzahl im Feld des ersten Operanden (D1(L1,B1)) wird mit der gepackten Dezimalzahl im Feld des zweiten Operanden (D2(L2,B2)) vorzeichengerecht verglichen und die Anzeige gemäß dem Vergleichsergebnis gesetzt.

Beide Operanden werden auf korrektes gepacktes Format geprüft, im Fehlerfalle erfolgt eine Programmunterbrechung wegen Datenfehlers.

Dezimal-Überlauf kann nicht auftreten.

Anzeige

0~Equal	Operand1 = Operand2
1~Low	Operand1 < Operand2
2~High	Operand1 > Operand2
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	Lesezugriff auf Operand1 oder Operand2 unmöglich.
Datenfehler	X'60'	Inkorrektes Format eines Operanden

Programmierhinweise

- Beide Operanden werden als Ganzzahlen verarbeitet.
- Beide Operanden bleiben stets unverändert.
- Die Verarbeitung der Operanden geschieht von rechts nach links.
- Unterschiedliche Vorzeichencodierungen, die aber gleiche Bedeutung haben, z.B. C_{16} und F_{16}, werden beim Vergleich gemäß ihrer Bedeutung behandelt.
- Der Vergleich einer negativen mit einer positiven Null ergibt die Anzeige 0~Equal.
- Die beiden Operanden dürfen sich überlappen, aber dann müssen die Adressen ihrer niedrigstwertigen Byte gleich sein (D1(B1)+L1-1=D2(B2)+L2-1), andernfalls erfolgt eine Programmunterbrechung wegen Datenfehlers.

Beispiele

Die folgenden Beispiele von CP-Befehlen ergeben folgende Ergebnisse:

CFIELD	Beispiel-Befehl	Anzeige
PL1’+0’	CP CFIELD,=PL16’-0’	0
PL1’+1’	CP CFIELD,=PL1’2’	1
PL1’+1’	CP CFIELD,=PL16’-2’	2

Im ersten Beispiel wird eine positive Null mit einer negativen Null verglichen, die überdies verschieden lang sind: das Vergleichsergebnis lautet dennoch "gleich". Ebenso würde der Vergleich z.B. von X’1B’ mit X’001D’ die Anzeige auf 0~Equal setzen.

Divide Decimal

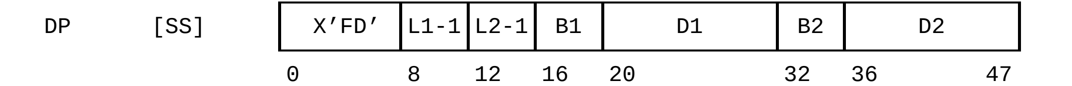
Funktion

Der Befehl DP dividiert zwei gepackte Dezimalzahlen. Der Quotient und Divisionsrest ersetzen den Dividenden.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
*	DP	D1(L1,B1),D2(L2,B2)	L2 < L1 ≤ 16 und 1 ≤ L2 ≤ Min(8, L1-1)

Maschinenformat



Beschreibung

Der Befehl DP dividiert vorzeichengerecht den Dividenden (D1(L1,B1)) durch den Divisor (D2(L2,B2)) und erzeugt den (ganzzahligen Teil des) Quotienten sowie den Divisionsrest als gepackte Dezimalzahlen im Feld des Dividenden.

Die Länge des Dividenden (L1) muß größer sein als die des Divisors (L2), andernfalls erfolgt eine Programmunterbrechung wegen Adreßfehlers (L2 < L1 ≤ 16). Mindestens die erste Dezimalstelle des Dividenden muß $=0_{16}$ sein, sonst erfolgt eine Programmunterbrechung wegen Divisionsfehlers.

Die Länge des Divisors (L2) muß kleiner sein als die des Dividenden (L1) und darf nicht größer sein als 8 Byte, andernfalls erfolgt eine Programmunterbrechung wegen Adreßfehlers. Der Divisor kann demnach höchstens 15 Dezimalstellen umfassen (L2 ≤ Min (L1-1,8)).

Der Quotient wird L1-L2 Byte lang und wird als gepackte Dezimalzahl (Ganzzahl) in die linken L1-L2 Byte des Felds des Dividenden gespeichert; der Divisionsrest wird L2 Byte lang und wird als gepackte Dezimalzahl (Ganzzahl) in die rechten L2 Byte des Felds des Dividenden gespeichert. Quotient und Divisionsrest ersetzen somit vollständig den Dividenden.

Beide Operanden müssen gültige gepackte Dezimalzahlen darstellen, andernfalls erfolgt eine Programmunterbrechung wegen Datenfehlers.

Das Vorzeichen des Quotienten wird nach den üblichen algebraischen Regeln gebildet, auch wenn der Dividend =0 ist; der Divisionsrest hat stets das Vorzeichen des Dividenden, auch wenn der Divisionsrest =0 ist. Ein positives Vorzeichen wird als C_{16}, ein negatives Vorzeichen wird als D_{16} dargestellt.

Wenn der Divisor =0 ist oder der Quotient länger ist als L1-L2 Byte, findet eine Programmunterbrechung wegen Divisionsfehlers statt (nicht eine Programmunterbrechung wegen Dezimal-Überlaufs). Dies gilt auch für den Fall, daß Dividend *und* Divisor beide =0 sind. Bei Divisionsfehler bleiben die Eingangsoperanden unverändert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.
Adreßfehler	X’5C	L1 oder L2 falsch
Datenfehler	X’60’	Inkorrektes Format eines Operanden
Divisionsfehler	X’68’	Divisor =0 oder Quotient zu groß

Programmierhinweise

- Alle Operanden (Dividend, Divisor, Quotient und Rest) werden als Ganzzahlen interpretiert.
- Der Quotient kann höchstens 15 Byte lang werden, also höchstens 29 Dezimalstellen umfassen.
- Dividend und Divisor dürfen sich überlappen, aber dann müssen die Adressen ihrer niedrigstwertigen Byte gleich sein (D1(B1)+L1-1=D2(B2)+L2-1), andernfalls erfolgt eine Programmunterbrechung wegen Datenfehlers; außerdem muß L1>L2, also D1(B1)≠D2(B2) gelten.
- Der Dividend muß wenigstens ein führendes Halbbyte mit dem Wert 0_{16} haben. Dies ist eine notwendige, nicht aber hinreichende Bedingung dafür, daß kein Divisionsfehler auftritt.

– Die folgende Befehlsfolge ist äquivalent der Prüfung auf Divisionsfehler, die innerhalb von DP abläuft:

Name	Operation	Operanden
	MVO CLC BNH	TEMP(L’DIVISOR+1),DIVISOR TEMP(L’DIVISOR),DIVIDEND DIVERROR

Diese Befehlsfolge kann man vor einem DP-Befehl ausführen, um sicherzustellen, daß keine Programmunterbrechung wegen Divisionsfehlers auftritt. Benötigt wird dafür ein temporäres Feld TEMP, das (L’DIVISOR + 1) Byte lang ist.

Beispiele

Die folgenden Beispiele ergeben jeweils nach der Ausführung von

Name	Operation	Operanden
	. DP .	DFIELD,DIVISOR

folgende Resultate:

DFIELD vorher Dividend	DIVISOR Divisor	DFIELD nachher Quotient	 Divisionsrest
PL5’1001’	PL2’10’	PL3’+100’	PL2’+1’
PL5’1001’	PL2’-10’	PL3’-100’	PL2’+1’
PL5’-1001’	PL2’-10’	PL3’+100’	PL2’-1’
PL5’1000’	PL2’-10’	PL3’-100’	PL2’+0’
PL3’-1000’	P’1’	unverändert, Divisionsfehler	
PL8’-1000’	PL8’10’	unverändert, Adreßfehler	

Im vorletzten Beispiel entsteht Divisionsfehler, weil der Quotient (P’-1000’) zu seiner Speicherung 3 Byte benötigen würde, aber nur 2 Byte zur Verfügung stehen; das dritte Byte ist für den Divisionsrest reserviert.

Im letzten Beispiel entsteht Adreßfehler, weil die Bedingung
$L2=8 \leq Min(L1-1,8)=Min(7,8)=7$ verletzt ist.

Edit

Funktion

Die Befehle ED (Edit) und EDMK (Edit and Mark) bereiten eine oder mehrere gepackte Dezimalzahlen in eine druckgerechte Form auf. Der Befehl EDMK trägt zusätzlich die Adresse der höchstwertigen signifikanten Dezimalstelle ins Mehrzweckregister 1 ein. Die Anzeige wird gemäß dem Wert der letzten aufbereiteten Dezimalzahl gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	ED EDMK	D1(L,B1),D2(B2) D1(L,B1),D2(B2)	1 ≤ L ≤ 256 1 ≤ L ≤ 256

Maschinenformate

		X'DE'	L-1	B1	D1	B2	D2
ED	[SS]	X'DE'	L-1	B1	D1	B2	D2
EDMK	[SS]	X'DF'	L-1	B1	D1	B2	D2
		0	8	16	20	32	36 47

Beschreibung

Die Befehle ED und EDMK bereiten eine oder mehrere aufeinanderfolgende gepackte Dezimalzahlen in eine druckgerechte Form auf. Die Aufbereitung geschieht unter der Steuerung einer Maske.

Der Befehl EDMK leistet das Gleiche wie der Befehl ED, er "markiert" jedoch zusätzlich die höchstwertige signifikante Dezimalstelle im Empfangsfeld, indem er deren Adresse ins Mehrzweckregister 1 einträgt.

Der mit D1(B1) adressierte Operand1 dient zwei Zwecken: Er enthält vor der Befehlsausführung die Aufbereitungs-Maske und nach der Befehlsausführung das aufbereitete Ergebnis. Je nach diesem Zweck wird der Operand1 deshalb entweder "Maske" oder "Empfangsfeld" genannt. In der Maske darf jedes beliebiges Zeichen stehen, jedoch haben drei Zeichen eine spezielle Bedeutung als Steuerzeichen. Es sind dies die Codierungen X’20’ (Ziffernselektor), X’21’ (Signifikanzstarter) und X’22’ (Feldtrenner).

Der Operand2 (D2(B2)) ist das "Sendefeld". Es muß eine oder mehrere aufeinanderfolgende gepackte Dezimalzahlen enthalten.

Wenn sich die beiden Operanden überlappen, entstehen unvorhersehbare Ergebnisse.

Beide Operanden werden von links nach rechts verarbeitet. Die Maske (und das Empfangsfeld) werden byteweise, das Sendefeld wird halbbyteweise verarbeitet.

Jedes Zeichen in der Maske wird während der Befehlsausführung entweder durch eine entpackte Dezimalziffer des Sendefeldes oder durch das Füllzeichen (siehe unten) ersetzt oder es bleibt unverändert. Welche dieser Möglichkeiten tatsächlich genutzt wird, hängt ab

- vom Maskenzeichen,
- von der Schalterstellung des Signifikanz-Indikators,
- vom Wert der Dezimalziffer, die dem Maskenzeichen im Sendefeld gegenüberliegt (=0 oder ≠0).

Maskenzeichen

Es gibt folgende Maskenzeichen:

Maskenzeichen	Codierung
Ziffernselektor	X'20'
Signifikanzstarter	X'21'
Feldtrenner	X'22'
Textzeichen	jede andere

Das Auftreten entweder eines Ziffernselektors oder eines Signifikanzstarters in der Maske bewirkt, daß die nächste Dezimalziffer des Sendefeldes gelesen wird. Je nach deren Wert sowie der Stellung des Signifikanz-Indikators wird entweder diese Dezimalziffer - entpackt und mit der Zone F_{16} versehen - oder das Füllzeichen anstelle des Maskenzeichens in das Empfangsfeld eingetragen.

Der Feldtrenner bestimmt, daß eine neue Dezimalzahl des Sendefeldes beginnt, wenn in einem Befehl ED oder EDMK mehrere Dezimalzahlen aufbereitet werden sollen. Der Feldtrenner wird immer durch das Füllzeichen ersetzt und bewirkt, daß der Signifikanz-Indikator in die Stellung "aus" gebracht wird.

Alle Textzeichen in der Maske bleiben je nach der Stellung "aus" bzw. "ein" des Signifikanz-Indikators entweder unverändert oder werden durch das Füllzeichen ersetzt.

Füllzeichen

Das erste Zeichen in der Maske, d.h. das Byte an der Adresse D1(B1), wird als Füllzeichen verwendet. Als Füllzeichen kann jedes beliebige Zeichen gewählt werden, auch der Ziffernselektor, der Signifikanzstarter oder der Feldtrenner.

Das Füllzeichen wird in Abhängigkeit von der Stellung des Signifikanz-Indikators, vom Maskenzeichen und dem Wert der ihm gegenüberliegenden Sendefeldziffer in das Empfangsfeld eingesetzt. Die Details sind in der Tabelle im Abschnitt Zusammenfassung dargestellt.

Das Füllzeichen wird zu Beginn der Befehlsausführung zwischengespeichert und steht vom Zwischenspeicher aus während der gesamten Befehlsausführung zur Verfügung, auch wenn es selbst im Empfangsfeld ersetzt wird. Erst nach der Zwischenspeicherung des Füllzeichens beginnt die Interpretation der Maskenzeichen, beginnend beim Füllzeichen selbst.

Das Füllzeichen wird nur dann selbst ersetzt, wenn es ein Ziffernselektor oder Signifikanzstarter ist und wenn ihm eine Dezimalziffer $\neq 0$ gegenüberliegt.

Sendefeldziffern

Bei jedem Auftreten eines Ziffernselektors oder Signifikanzstarters in der Maske wird die nächste Dezimalziffer des Sendefeldes gelesen und der Signifikanz-Indikator geprüft. Abhängig davon wird die Dezimalziffer entweder in das rechte Halbbyte des Empfangsfeldes gespeichert und in das linke Halbbyte die Zone F_{16} eingetragen oder es wird die Dezimalziffer übergangen und in das Empfangsfeld das Füllzeichen gespeichert.

Das Sendefeld wird halbbyteweise von links nach rechts verarbeitet. Alle linken Halbbyte müssen Dezimalziffern, also Sedezimalziffern $\leq 9_{16}$ enthalten, andernfalls erfolgt eine Programmunterbrechung wegen Datenfehlers. Jedesmal, wenn ein linkes Halbbyte gelesen wird, wird auch gleich das rechte Halbbyte daraufhin geprüft, ob es ein Vorzeichen, d.h. eine Sedezimalziffer $\geq A_{16}$ enthält. Wenn ja, wird (nachdem ggf. der Signifikanz-Indikator umgeschaltet wurde) dafür gesorgt, daß beim nächsten Ziffernselektor oder Signifikanzstarter das nachfolgende linke Halbbyte gelesen wird, andernfalls bleibt dafür das rechte Halbbyte verfügbar.

Signifikanz-Indikator

Der Signifikanz-Indikator ist ein interner Ein-Aus-Schalter. Wenn er sich in der Stellung "ein" befindet, herrscht "Signifikanz": Ziffernselektoren und Signifikanzstarter in der Maske werden dann durch ihre gegenüberliegenden Dezimalziffern ersetzt und Textzeichen bleiben unverändert. Wenn sich der Signifikanz-Indikator in der Stellung "aus" befindet, herrscht "Nicht-Signifikanz": Ziffernselektoren, Signifikanzstarter und Textzeichen werden dann durch das Füllzeichen ersetzt.

Der Signifikanz-Indikator kennzeichnet auch durch seine Ein-Aus-Stellung, ob die aufzubereitende Dezimalzahl ein Minus- oder Plus-Vorzeichen enthalten hat und bestimmt damit u.a. die Anzeige der Befehle ED und EDMK.

Der Signifikanz-Indikator wird in die Stellung "aus" geschaltet,

- zu Beginn der Befehlsausführung oder
- wenn ein Feldtrenner auftritt oder
- wenn einer Dezimalziffer in einem linken Halbbyte des Sendefelds ein Plus-Vorzeichen (A_{16}, C_{16}, E_{16} oder F_{16}) im zugehörigen rechten Halbbyte folgt.

Der Signifikanz-Indikator wird in die Stellung "ein" geschaltet, wenn er sich in der Stellung "aus" befindet und

- einem Signifikanzstarter eine Sendefeldziffer gegenüberliegt, der kein Plus-Vorzeichen folgt oder
- einem Ziffernselektor eine Sendefeldziffer gegenüberliegt, die $\neq 0$ ist und der kein Plus-Vorzeichen folgt.

In allen anderen Fällen wird der Signifikanz-Indikator nicht verändert (siehe dazu auch die zusammenfassende Tabelle weiter unten).

Empfangsfeld-Zeichen

Das Resultat der Aufbereitung durch die Befehle ED und EDMK wird in das Empfangsfeld gespeichert und ersetzt die Maske: es hat deren Länge (von L Byte). Es besteht aus Textzeichen, Füllzeichen und gezonten Sendefeldziffern.

Ein Textzeichen in der Maske bleibt entweder unverändert oder wird durch das Füllzeichen ersetzt, je nachdem ob bei seinem Auftreten sich der Signifikanz-Indikator in der Stellung "ein" bzw. in der Stellung "aus" befindet.

Ein Ziffernselektor oder Signifikanzstarter in der Maske wird durch das Füllzeichen ersetzt, wenn sich der Signifikanz-Indikator in der Stellung "aus" befindet und die gegenüberliegende Sendefeldziffer =0 ist. Dagegen wird ein Ziffernselektor oder Signifikanzstarter in der Maske durch die gegenüberliegende gezonte Sendefeldziffer ersetzt, wenn diese $\neq 0$ ist oder sich der Signifikanz-Indikator in der Stellung "ein" befindet.

Zusammenfassung

Die folgende Tabelle faßt die Funktionen der einzelnen Maskenzeichen zusammen. Die linken 4 Spalten zeigen die 4 möglichen Bedingungs-Konstellationen, die rechten 2 Spalten zeigen das jeweilige Ergebniszeichen im Empfangsfeld und die Stellung des Signifikanz-Indikators danach.

Wirkung der Maskenzeichen von ED und EDMK					
Bedingungen				Ergebnis	
Masken-zeichen	Signifikanz-Indikator vorher	Sendefeld-Ziffer	folgt Plus-Vorzeichen ?	Empfangs-feld-Zeichen	Signifikanz-Indikator nachher
Ziffern-selektor (X’20’)	aus aus aus ein ein	0 1...9 1...9 0...9 0...9	unerheblich nein ja nein ja	Füllzeichen SF-Ziffer SF-Ziffer SF-Ziffer SF-Ziffer	aus ein aus ein aus
Signifi-kanz-starter (X’21’)	aus aus aus aus ein ein	0 0 1...9 1...9 0...9 0...9	nein ja nein ja nein ja	Füllzeichen Füllzeichen SF-Ziffer SF-Ziffer SF-Ziffer SF-Ziffer	ein aus ein aus ein aus
Feld-trenner (X’22’)	unerheblich	unerheblich	unerheblich	Füll-zeichen	aus
Text-zeichen (jedes andere)	aus ein	unerheblich unerheblich	unerheblich unerheblich	Füll-zeichen Text-zeichen	aus ein

Anzeige

0~Zero   Alle verarbeiteten Ziffern der zuletzt aufbereiteten Dezimalzahl waren $=0_{16}$ oder in der Maske für die letzte aufbereitete Dezimalzahl war weder ein Signifikanzstarter noch ein Ziffernselektor aufgetreten oder das letzte Zeichen der Maske war ein Feldtrenner.

1~Minus   Die zuletzt aufbereitete Dezimalzahl war ungleich 0 und der Signifikanz-Indikator befand sich zuletzt in der Stellung "ein".

2~Plus   Die zuletzt aufbereitete Dezimalzahl war ungleich 0 und der Signifikanz-Indikator befand sich zuletzt in der Stellung "aus".

3   Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.
Datenfehler	X’60’	Ein linkes Halbbyte im Sendefeld enthält keine Dezimalziffer

Zusätzliche Markierung bei EDMK

Der Befehl EDMK ist bezüglich des oben beschriebenen Ablaufs und seines Ergebnisses sowie bezüglich der resultierenden Anzeige identisch zum Befehl ED. EDMK speichert jedoch zusätzlich die Adresse der höchstwertigen Dezimalziffer ≠0 im Empfangsfeld ins Mehrzweckregister 1, wenn sich bei deren Speicherung der Signifikanz-Indikator vorher in der Stellung "ein" befand. Im Falle der Aufbereitung mehrerer Dezimalzahlen mit einem Befehl EDMK gilt dies für die letzte Dezimalzahl, bei der die Signifikanz auf diese Weise eingeschaltet wurde. Wenn die Signifikanz überhaupt nicht oder durch den Signifikanzstarter eingeschaltet wurde, wird das Mehrzweckregister 1 nicht verändert.

Im 24-Bit-Adressierungsmodus wird die Adresse in die Bitstellen 8 bis 31, im 31-Bit-Adressierungsmodus wird die Adresse in die Bitstellen 1 bis 31 des Mehrzweckregisters 1 eingetragen. Die Bitstellen 0 bis 7 bzw. die Bitstelle 0 bleiben unverändert.

Programmierhinweise

- Die Befehle ED und EDMK sind vorgesehen für solche Fälle, bei denen aufbereitete Dezimalzahlen zusätzliche Zeichen wie z.B. Tausenderpunkte, Dezimalkommas oder Währungszeichen enthalten sollen oder bei denen führende Nullen durch ein Füllzeichen, z.B. durch ein Leerzeichen oder einen "Schutzstern" ersetzt sein sollen (siehe Beispiele).
- Der Befehl EDMK erleichtert das Einfügen eines Vorzeichens unmittelbar vor die höchstwertige von Null verschiedene Dezimalziffer der letzten aufbereiteten Dezimalzahl. Wenn für die Aufbereitung dieser Dezimalzahl in der Maske ein Signifikanzstarter verwendet wurde, ist zu empfehlen, vor dem Aufruf von EDMK dessen Adresse ins Mehrzweckregister 1 zu laden, allerdings um 1 erhöht. Dann enthält nämlich das Register 1 nach dem Befehl EDMK sowohl bei Signifikanz-Einschaltung durch den Signifikanzstarter als auch bei der Signifikanz-Einschaltung durch die höchstwertige Dezimalziffer ≠0 die Adresse der höchstwertigen gezonten Dezimalziffer und es zeigt demzufolge das um Eins verminderte Register 1 auf die Position des gleitenden Vorzeichens (siehe dazu auch Beispiel (3)).
- Der Befehl EDMK verändert (bei bestimmten Datenkonstellationen) das Mehrzweckregister 1, obwohl dieses nicht im Befehl angegeben wird.

– Weil das Mehrzweckregister 1 bei EDMK nicht in jedem Falle geändert wird und auch dann, wenn es geändert wird, nicht vollständig ersetzt wird, empfiehlt es sich, dieses Register 1 in jedem Falle, nicht nur im o.a. Fall, *vor* EDMK auf einen sinnvollen Wert zu setzen.

Beispiele

```
Name        | Operation | Operanden
------------+-----------+------------------------------------------------------
            | MVC       | DFIELD1,MASKE
Beispiel1   | ED        | DFIELD1,=P'123456789'
*           |           |                   DFIELD1 nachher: 1'234.567,89
*           |           |                   Anzeige: 2~Plus
MASKE       | DC        | C' '              Füllzeichen
            | DC        | X'20'             Ziffernselektor
            | DC        | C''''             Apostroph
            | DC        | 3X'20'            3 Ziffernselektoren
            | DC        | C'.'              Punkt
            | DC        | X'202120'         Ziffernselektor,Signifikanzstar-
*           |           |                   ter, Ziffernselektor
            | DC        | C','              Komma
            | DC        | 2X'20'            2 Ziffernselektoren
            | .         |
Beispiel2   | ED        | DFIELD2(11),SFIELD
*           |           |                   DFIELD2 nachher: ***1002***3
*           |           |                   Anzeige 1~Minus
DFIELD2     | DC        | C'*'              Füllzeichen
            | DC        | 2X'202120'        erste 2 Masken
            | DC        | X'22'             Feldtrenner
            | DC        | X'202120'         dritte Maske
SFIELD      | DC        | PL2'-1,-2,-3'     3 aufzubereitende Zahlen
            | .         |
Beispiel3   | MVC       | DFIELD3(7),=X'402021206B2020'
*           |           |                   X'40'=Leerzeichen, X'6B'=Punkt
            | LA        | 1,DFIELD3+3       Register 1 auf
*           |           |                   Signifikanzstarter +1
            | EDMK      | DFIELD3(7),ACCOUNT ACCOUNT = 3 Byte lange
            | BE        | READY             gepackte Dezimalzahl
            | BCTR      | 1,0
            | MVI       | 0(1),'+'
            | BH        | READY
            | MVI       | 0(1),'-'
READY       | EQU       | *
```

Wenn im Beispiel1 statt P’123456789’ als aufzubereitende Zahl PL5’-1’ angegeben wäre, würde die Zeichenfolge ␣␣␣␣␣␣␣␣␣␣0,01 entstehen.

Das Beispiel2 zeigt die Wirkung von Minus-Vorzeichen und Feldtrenner auf den Signifikanz-Indikator. Wenn die erste aufzubereitende Dezimalzahl statt PL2’-1’ PL2’+1’ lautet, entsteht in DFIELD2 die Zeichenfolge ***1**2***3, weil dann der Signifikanz-Indikator durch das Plus-Vorzeichen ausgeschaltet wird.

Das Beispiel3 zeigt eine Befehlsfolge zur Druckaufbereitung eines Geldbetrages, dem je nachdem, ob er >0, <0 oder =0 ist, entweder ein +, ein - oder kein Zeichen vorangestellt werden soll. So soll beispielsweise der ACCOUNT-Wert P’+123.45’ zu +123,45, der Wert PL3’-.12’ zu ␣␣-0,12 und der Wert PL3’0’ zu ␣␣␣0,00 aufbereitet werden. Die Befehlsfolge lädt vor EDMK das Mehrzweckregister 1 "vorsorglich" mit der Adresse des Zeichens, das dem Signifikanzstarter (X’21’) in der Maske folgt. Wenn sich die erste Nicht-Null in ACOUNT rechts vom Signifikanzstarter befindet, bleibt durch EDMK das Register 1 unverändert, andernfalls trägt EDMK deren Adresse ins Register 1 ein. In beiden Fällen enthält nach EDMK das Register 1 die Adresse der ersten gezonten Ziffer und folglich zeigt der mittels BCTR 1,0 um Eins reduzierte Register 1-Inhalt auf die Dezimalstelle, wo das "gleitende" + oder - oder, im Falle 0, nichts hingehört.

Multiply Decimal

Funktion

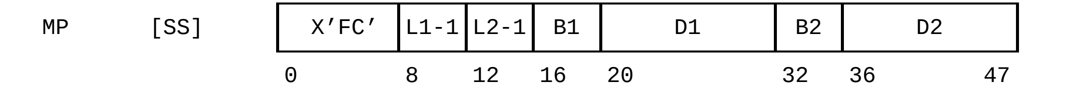
Der Befehl MP multipliziert vorzeichengerecht zwei gepackte Dezimalzahlen. Das Produkt ersetzt den ersten Operanden.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
*	MP	D1(L1,B1),D2(L2,B2)	L2 < L1 ≤ 16 und 1 < L2 ≤ Min(8, L1-1)

Maschinenformat

MP [SS]



Beschreibung

Der gepackte Multiplikator wird mit dem gepackten Multiplikanden vorzeichengerecht multipliziert. Das gepackte Produkt ersetzt den Multiplikanden.

Der Multiplikand und das Produkt sind mit D1(B1), der Multiplikator ist mit D2(B2) im Hauptspeicher adressiert. Die Länge des Multiplikators (L2) muß mindestens 1 und darf maximal 8 Byte betragen; sie muß außerdem kleiner sein als die Länge des Multiplikanden (1≤L2≤Min(L1-1,8)). Die Länge des Multiplikanden (L1) muß größer sein als die des Multiplikators (L2<L1≤16). Wenn diese Bedingungen nicht erfüllt sind, erfolgt eine Programmunterbrechung wegen Adreßfehlers.
Der Multiplikand muß mindestens soviele führende Byte mit dem Inhalt 00_{16} besitzen wie der Multiplikator lang ist, andernfalls erfolgt eine Programmunterbrechung wegen Datenfehlers. Eine solche erfolgt auch, wenn einer oder beide Operanden nicht korrekt gepackte Dezimalzahlen darstellen.

Das Vorzeichen des Produkts wird nach den algebraischen Regeln ermittelt, auch wenn ein oder beide Operanden =0 sind.

Ein Dezimal-Überlauf kann nicht auftreten.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich.
Adreßfehler	X’5C’	L1 oder L2 falsch
Datenfehler	X’60’	1. Inkorrektes Format eines Operanden 2. Der Multiplikand hat nicht wenigstens L2 führende Null-Byte

Programmierhinweise

- Beide Operanden werden als Ganzzahlen verarbeitet, das Produkt ist eine Ganzzahl.
- Im Produkt wird ein positives Vorzeichen durch C_{16}, ein negatives Vorzeichen durch D_{16} dargestellt.
- Multiplikand und Multiplikator dürfen sich überlappen, aber dann müssen die Adressen ihrer niedrigstwertigen Byte gleich sein (D1(B1)+L1-1= D2(B2)+L2-1), andernfalls erfolgt eine Programmunterbrechung wegen Datenfehlers; außerdem muß L1>L2, also D1(B1)<D2(B2) gelten und die oberen L2 Byte des Multiplikanden müssen $=00_{16}$ sein (siehe Beispiel).
- Ein Produkt =0 kann ein negatives Vorzeichen haben.
- Das Resultat eines MP-Befehls hat stets mindestens eine führende Null (0_{16}).
- Das Resultat eines MP-Befehls kann dem Betrage nach höchstens 10^{30}-2*10^{15}+1, also höchstens 29-stellig werden.

Beispiele

Die folgenden Beispiele von MP-Befehlen ergeben folgende Ergebnisse:

DFIELD vorher	Beispiel-Befehl	DFIELD nachher
PL2’-9’	MP DFIELD,=P’2’	PL2’-18’
PL2’-9’	MP DFIELD,=P’0’	PL2’-0’
PL2’-9’	MP DFIELD,DFIELD+1(1)	PL2’+81’

Man beachte, daß anstelle des dritten Beispiels der Befehl MP DFIELD,DFIELD nicht korrekt wäre, weil die Eingangsbedingung L2<L1 nicht erfüllt ist.

Subtract Decimal

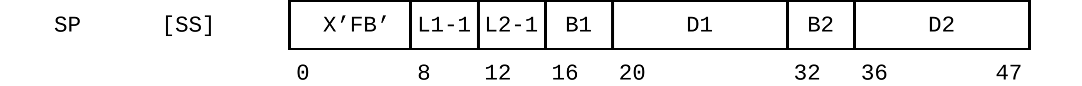
Funktion

Der Befehl SP subtrahiert zwei gepackte Dezimalzahlen. Die resultierende Differenz ersetzt den ersten Operanden.
Die Anzeige wird gemäß dem Wert der Differenz gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	SP	D1(L1,B1),D2(L2,B2)	1 ≤ L1,L2 ≤ 16

Maschinenformat



Beschreibung

Die gepackte Dezimalzahl im Feld des zweiten Operanden (D2(L2,B2)) wird von der gepackten Dezimalzahl im Feld des ersten Operanden (D1(L1,B1)) vorzeichengerecht subtrahiert; die gepackte Differenz ersetzt den ersten Operanden.

Der erste Operand und die Differenz haben die Länge L1 Byte, der zweite Operand hat die Länge L2 Byte, 1≤L1,L2≤16.

Beide Operanden werden auf korrektes gepacktes Format geprüft, im Fehlerfalle erfolgt eine Programmunterbrechung wegen Datenfehlers.

Ein Dezimal-Überlauf tritt auf, wenn das Resultat mehr signifikante Dezimalstellen besitzt als in das Feld des ersten Operanden passen. Der Befehl wird normal beendet, jedoch werden nur die 2L1-1 niedrigstwertigen Dezimalstellen der Differenz gespeichert und die höchstwertigen Dezimalstellen gehen verloren; die Anzeige wird dann auf 3~Overflow gesetzt. Wenn das Bit für Dezimal-Überlauf in der Programmaske auf 1 gesetzt ist (BS2000-Standard), erfolgt außerdem danach eine Programmunterbrechung.

Eine echte Differenz =0 hat stets ein positives Vorzeichen (C_{16}), jedoch kann eine Differenz =0, die durch Dezimal-Überlauf entstanden ist, auch ein negatives Vorzeichen haben (D_{16}).

Anzeige

0~Zero Die Differenz ist =0 (sie hat das Vorzeichen C_{16}).
1~Minus Die Differenz ist <0 (sie hat das Vorzeichen D_{16}).
2~Plus Die Differenz ist >0 (sie hat das Vorzeichen C_{16}).
3~Overflow Die Differenz hat mehr signifikante Dezimalstellen als in das Feld des ersten Operanden passen.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lese-Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich
Datenfehler	X’60’	Inkorrektes Format eines Operanden
Dezimal-Überlauf	X’74’	Differenz zu groß für den ersten Operanden

Programmierhinweise

- Beide Operanden werden als Ganzzahlen verarbeitet.
- Die Verarbeitung der Operanden erfolgt von rechts nach links.
- Die beiden Operanden dürfen sich überlappen, aber dann müssen die Adressen ihrer niedrigstwertigen Byte gleich sein (D1(B1)+L1-1 = D2(B2)+L2-1), andernfalls erfolgt eine Programmunterbrechung wegen Datenfehlers.
- Der zweite Operand wird nur verändert, wenn er sich mit dem ersten Operanden überlappt.
- Bei einem Dezimal-Überlauf hat das Resultat das Vorzeichen der korrekten Differenz, deshalb kann bei Dezimal-Überlauf das Resultat =0 ein negatives Vorzeichen haben.

Beispiele

Die folgenden Beispiele von SP-Befehlen ergeben folgende Ergebnisse:

DFIELD vorher	Beispiel-Befehl	DFIELD nachher	Anzeige
PL1'-2'	SP DFIELD,=P'-10'	PL1'+8'	2
PL1'-2'	SP DFIELD,=PL16'-2'	PL1'+0'	0
PL1'-2'	SP DFIELD,=P'-13'	PL1'+1'	3
PL1'-2'	SP DFIELD,DFIELD	PL1'+0'	0

Man beachte, daß der zweite Operand durchaus länger sein darf als der erste Operand (erste drei Beispiele). Dezimal-Überlauf tritt erst auf, wenn das Resultat des SP-Befehls zu lang ist, um im ersten Operanden gespeichert werden zu können.

Shift and Round Decimal

Funktion

Der Befehl SRP verschiebt eine gepackte Dezimalzahl um eine angegebene Anzahl von Dezimalstellen nach links oder rechts; im Falle von Rechts-Verschiebung wird die Dezimalzahl anschließend gemäß einer angegebenen Rundungsziffer gerundet.
Die Anzeige wird gemäß dem Wert des Resultats gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
	SRP	D1(L1,B1),D2(B2),I3	0 ≤ I3 ≤ 9
* oder auch:			
	SRP	D1(L1,B1),64-r,rz	1 ≤ r ≤ 32; 0 ≤ rz ≤ 9
	SRP	D1(L1,B1),l,rz	0 ≤ l ≤ 31; 0 ≤ rz ≤ 9

Darin bedeutet:

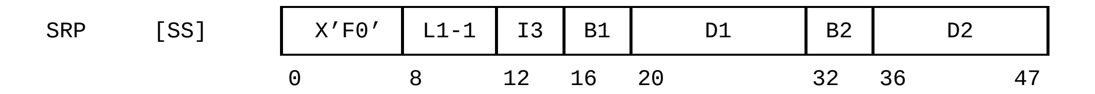
`r` die Anzahl nach rechts zu verschiebender Dezimalstellen,

`rz` die Rundungsziffer (≤9) und

`l` die Anzahl nach links zu verschiebender Dezimalstellen.

Auch bei Linksverschiebung muß im Assemblerformat der Direktoperand I3 angegeben sein, obwohl er bei der Befehlsausführung nicht berücksichtigt wird.

Maschinenformat



Beschreibung

Der mit D1(L1,B1) adressierte erste Operand muß eine gepackte Dezimalzahl der Länge L1 Byte darstellen (1 ≤ L1 ≤ 16). Diese Dezimalzahl wird in die Richtung und um die Anzahl von Dezimalstellen verschoben, die durch die Adresse D2(B2) bestimmt sind.
Der Direktoperand I3 muß eine Dezimalziffer, d.h. ≤9 sein und dient im Falle von Rechts-Verschiebung als Rundungsziffer für die abschließende Rundung.

Die durch D2(B2) bestimmte Adresse wird nicht als Datenadresse verwendet; stattdessen bilden die rechten 6 Binärstellen dieser Adresse (Bit 26-31) die Verschiebeinformation: Wenn das höchstwertige Bit dieser Binärzahl =0 ist, (d.h. wenn das Bit 26 =0 ist), wird nach links verschoben und zwar um so viele Dezimalstellen, wie die Binärzahl angibt. Andernfalls, wenn das höchstwertige Bit der Binärzahl =1 ist, wird nach rechts verschoben und zwar um so viele Dezimalstellen wie das Zweierkomplement der Binärzahl angibt.

Sowohl bei Links- wie bei Rechts-Verschiebung bleibt die Vorzeichenstelle des ersten Operanden unverändert. Freiwerdende Ziffernpositionen werden mit 0_{16} aufgefüllt, bei einer Links-Verschiebung von rechts, bei einer Rechts-Verschiebung von links.

Wenn durch Links-Verschiebung eine signifikante Dezimalziffer verloren geht, tritt Dezimal-Überlauf ein. Die Anzeige wird dann auf 3~Overflow gesetzt und es erfolgt eine Programmunterbrechung, wenn in der Programmaske das Bit für den Dezimal-Überlauf =1 gesetzt ist (BS2000-Standard).

Rundung:
Im Falle von Rechts-Verschiebung wird abschließend die verschobene Dezimalzahl gerundet. Dazu wird die zuletzt hinausgeschobene Dezimalziffer zum Direktoperanden I3 (Rundungsziffer) addiert und eine etwaige Überlauf-Eins auf die verschobene Dezimalzahl gemäß deren Vorzeichen addiert.

Der erste Operand wird auf korrektes gepacktes Format geprüft. Der Wert von I3 muß eine korrekte Dezimalziffer, d.h. ≤ 9 sein, auch wenn nach links verschoben wird. Im Fehlerfalle erfolgt eine Programmunterbrechung wegen Datenfehlers.

Anzeige

0~Zero	Resultat ist = 0 (+0 oder -0).
1~Minus	Resultat ist < 0.
2~Plus	Resultat ist > 0.
3~Overflow	Dezimal-Überlauf.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lese-Schreibzugriff auf Operand1 unmöglich
Datenfehler	X’60’	1. Inkorrektes Format des ersten Operanden 2. Rundungsziffer (I3) nicht ≤ 9
Dezimal-Überlauf	X’74’	Verlust signifikanter Dezimalstellen

Programmierhinweise

- Die Verschiebe-Information (Richtung und Anzahl) wird aus den 6 niedrigstwertigen Bit der Adresse D2(B2) gewonnen. Diese 6 Bit bilden eine Binärzahl b im Wertebereich $0 \leq b \leq 63$. Wenn $b \leq 31$ ist, wird nach links um b Dezimalstellen verschoben, wenn $32 \leq b \leq 63$ ist, wird nach rechts um 64-b Dezimalstellen verschoben.

  Es ergeben sich folgende Grenzwerte:

6 niedrigstwertige Bit von D2(B2)	Binärzahl b	Verschiebe-Richtung	Verschiebe-Zahl
000000	0	keine Verschiebung	
000001	1	links	1 Dezimalstelle
011111	31	links	31 Dezimalstellen
100000	32	rechts	32 Dezimalstellen
111111	63	rechts	1 Dezimalstelle

- Wenn B2=0 ist, wird die Verschiebe-Information allein aus den 6 niedrigstwertigen Bit des D2-Feldes gewonnen. In diesem Fall braucht im Assemblerformat "(B2)" nicht geschrieben zu werden.
- Wenn gerundet wird, wird bei positiven Zahlen nach oben und bei negativen Zahlen nach unten gerundet.
- Die übliche, kommerzielle Rundung wird durch die Rundungsziffer 5 erreicht.
- Eine reine Rechts-Verschiebung wird durch eine Rundungsziffer 0 erreicht.
- Der Maximalwert 32 für eine Rechts-Verschiebung reicht aus, um die längstmögliche Dezimalzahl zu "nullen" (31 würde auch genügen).
- Auch bei Linksverschiebung muß im Assemblerformat ein dritter Operand I3 angegeben und $\leq 9_{16}$ sein, obwohl er bei der Befehlsausführung nicht berücksichtigt wird.
- Die Multiplikation einer gepackten Dezimalzahl GZAHL mit einer *variablen* Zehnerpotenz 10^{x+k} wird erreicht, indem man die Variable x in das Mehrzweckregister B1 und die Konstante k in das D1-Feld eines SRP einsetzt, also z.B. schreibt:

Name	Operation	Operanden
	LH SRP	6,=H'x' GZAHL,k(6),0

  Das Besondere daran ist, daß dies auch funktioniert, wenn x+k negativ ist, also durch $10^{-(x+k)}$ dividiert werden soll.

Beispiele

Die folgenden Beispiele von SRP-Befehlen ergeben:

DFIELD vorher	Beispiel-Befehl		DFIELD nachher	Anzeige
PL2’995’	SRP	DFIELD,64-1,5	PL2’100’	2
PL2’994’	SRP	DFIELD,64-1,5	PL2’99’	2
PL2’-995	SRP	DFIELD,64-1,5	PL2’-100’	1
PL2’-1’	SRP	DFIELD,3,0	PL2’-0’	3
PL2’-1’	SRP	DFIELD,1,0	PL2’-10’	1
PL2’-1’	SRP	DFIELD,64-1,9	PL2’-1’	1

Zero and Add

Funktion

Der Befehl ZAP überträgt eine gepackte Dezimalzahl in den angegebenen Hauptspeicherbereich. Der Befehl entspricht dem Befehl AP, wenn dessen erster Operand =0 ist. Die Anzeige wird gemäß dem Wert der übertragenen Dezimalzahl gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	ZAP	D1(L1,B1),D2(L2,B2)	1 ≤ L1,L2 ≤ 16

Maschinenformat

ZAP [SS]

X'F8'	L1-1	L2-1	B1	D1	B2	D2
0	8	12	16	20	32	36 47

Beschreibung

Mit D1(L1,B1) wird das Empfangsfeld, mit D2(L2,B2) wird das Sendefeld adressiert (1≤L1,L2≤16).

Nur das Sendefeld wird auf korrektes gepacktes Format geprüft, im Fehlerfalle erfolgt eine Programmunterbrechung wegen Datenfehlers.

Ein Dezimal-Überlauf tritt auf, wenn das Sendefeld mehr signifikante Dezimalstellen aufweist als in das Empfangsfeld passen; in diesem Fall wird der Befehl normal beendet, es werden jedoch nur die 2L1-1 niedrigstwertigen Dezimalstellen des Sendefelds übertragen und die höchstwertigen Dezimalstellen gehen verloren. Die Anzeige wird auf 3~Overflow gesetzt. Wenn das Bit für Dezimal-Überlauf in der Programmaske auf 1 gesetzt ist (BS2000-Standard), erfolgt außerdem danach eine Programmunterbrechung.

Nach der Befehlsausführung hat eine echte Null im Sendefeld stets ein positives Vorzeichen (C_{16}), jedoch kann ein Resultat =0, das durch Dezimal-Überlauf entstanden ist, auch ein negatives Vorzeichen haben.

Anzeige

0~Zero	Empfangsfeld = 0 (Vorzeichen C_{16}).
1~Minus	Empfangsfeld < 0 (Vorzeichen D_{16}).
2~Plus	Empfangsfeld > 0 (Vorzeichen C_{16}).
3~Overflow	Sendefeld hat mehr signifikante Dezimalstellen als in das Empfangsfeld passen.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Schreibzugriff auf Operand1 oder Lesezugriff auf Operand2 unmöglich
Datenfehler	X’60’	Inkorrektes Format des Sendefelds
Dezimal-Überlauf	X’74’	zweiter Operand zu groß

Programmierhinweise

– Das Sendefeld wird nur geändert, wenn es sich mit dem Empfangsfeld überlappt.

– Die Übertragung geschieht von rechts nach links.

– Die beiden Operanden dürfen sich überlappen. Ein korrektes Ergebnis ist aber nur dann zu erwarten, wenn das niedrigstwertige Byte des Sendefelds nicht rechts vom niedrigstwertigen Byte des Empfangsfelds liegt, wenn also D1(B1)+L1-1 ≤ D2(B2)+L2-1 ist.

– Das Vorzeichen des Empfangsfelds wird auch dann $=C_{16}$ bzw. $=D_{16}$ gesetzt, wenn es im Sendefeld anders codiert ist.

Beispiele

FIELD vorher	Beispiel-Befehl	FIELD nachher	Anzeige
ohne Belang	ZAP FIELD(1),=PL1’-1’	PL1’-1’	1
ohne Belang	ZAP FIELD(1),=PL16’0’	PL1’+0’	0
ohne Belang	ZAP FIELD(1),=PL8’-10’	PL1’-0’	3

Man beachte, daß der Dezimal-Überlauf im dritten Beispiel nicht deshalb auftritt, weil der zweite Operand länger ist als der erste Operand, sondern weil sein Wert -10 nicht in den ersten Operanden paßt. Bei Dezimal-Überlauf kann es - wie hier - geschehen, daß eine resultierende Null ein negatives Vorzeichen bekommt.

5 Gleitpunktbefehle

Überblick

Die Gleitpunktbefehle dienen der Bearbeitung von Zahlen mit großem Wertebereich.

Es gibt Gleitpunktbefehle zum Laden, Speichern, Addieren, Subtrahieren, Multiplizieren, Dividieren, Vergleich sowie zur Vorzeichenbehandlung. Die Befehle verarbeiten drei Formate von Gleitpunktzahlen: das kurze, das lange und das erweiterte Format.

Die meisten Gleitpunktbefehle verarbeiten zwei Gleitpunktzahlen. Diese befinden sich entweder beide in Gleitpunktregistern oder es befindet sich die eine von ihnen im Hauptspeicher und die andere in einem Gleitpunktregister.

Die meisten Gleitpunktbefehle erzeugen normalisierte Ergebnisse, die die höchstmögliche Genauigkeit repräsentieren; für Addition und Subtraktion gibt es jedoch auch Befehle, die nicht-normalisierte Ergebnisse liefern, weil dies in manchen Anwendungsfällen, z.B. bei Zwischenergebnissen wünschenwert ist.

Gleitpunktzahlen (Vorzeichen, Charakteristik, Mantisse)

Jede Gleitpunktzahl besteht aus dem Vorzeichen, der Charakteristik und der Mantisse. Das Vorzeichen ist eine 1 Bit-Zahl. Eine 0 steht für ein positives, eine 1 steht für ein negatives Vorzeichen.

Die Charakteristik ist eine vorzeichenlose 7 Bit-Zahl; sie repräsentiert einen Exponenten zur Basis 16. Der Exponent selbst ergibt sich durch Subtraktion von 64 von der Charakteristik. Der Wertebereich der Charakteristik reicht von 0 bis 127, der des Exponenten reicht von -64 bis +63.
Die Mantisse ist ein Sedezimalbruch aus 6, 14 oder 28 Sedezimalstellen je nach Format (s.u.); der (implizite) Sedezimalpunkt dieses Bruchs befindet sich links vor der höchstwertigen Sedezimalstelle.

Der Wert einer Gleitpunktzahl ergibt sich aus Vorzeichen und dem Produkt aus Mantisse und der mit dem Exponenten potenzierten Basis 16:

Wert einer Gleitpunktzahl $= (-1)^{\text{Vorzeichen}} * \text{Mantisse} * 16^{\text{Exponent}}$
$= (-1)^{\text{Vorzeichen}} * \text{Mantisse} * 16^{\text{Charakteristik-64}}$

Exponenten-Überlauf und -Unterlauf

Wenn bei einer Gleitpunkt-Operation der resultierende Exponent kleiner als -64 (d.h. die Charakteristik kleiner als 0) wird, entsteht **Exponenten-Unterlauf**. Die Operation wird zu Ende geführt. Wenn dann das Bit für Exponenten-Unterlauf in der Programmaske =1 ist (BS2000-Standard), erfolgt eine Programmunterbrechung; Mantisse und Vorzeichen sind korrekt, aber die Charakteristik des Ergebnisses um 128 zu groß. Wenn jedoch bei Exponenten-Unterlauf das zugehörige Bit in der Programmaske =0 ist, erfolgt keine Programm-Unterbrechung; stattdessen wird als Ergebnis eine sog. echte Null erzeugt.

Wenn bei einer Gleitpunkt-Operation der resultierende Exponent größer als 63 (d.h. die Charakteristik größer als 127) wird, entsteht **Exponenten-Überlauf**. Die Operation wird zu Ende geführt und es erfolgt dann eine Programmunterbrechung. (Diese Programmunterbrechung erfolgt in jedem Falle: es gibt in der Programmaske *kein* Bit für Exponenten-Überlauf). Die Charakteristik des Ergebnisses ist um 128 zu klein. Die Mantisse und das Vorzeichen sind aber korrekt.

Behandlung der Null, Signifikanz

Wenn die Mantisse einer Addition, Subtraktion, Multiplikation oder Division =0 wird, so wird das Vorzeichen stets positiv gesetzt, dagegen hängt das Vorzeichen einer Mantisse =0 bei anderen Operationen vom Vorzeichen des oder der Eingangsoperanden ab (ebenso wie bei einem Ergebnis mit von 0 verschiedener Mantisse).

Wenn als Zwischenergebnis einer Gleitpunkt-Addition oder -Subtraktion eine Mantisse zustande kommt, deren Sedezimalstellen alle $=0_{16}$ sind, entsteht **Signifikanz**. Der Befehl wird zu Ende geführt. Wenn dann das Bit für Signifikanz in der Programmaske =1 ist, erfolgt eine Programmunterbrechung, wobei die Charakteristik korrekt ist, aber das Vorzeichen und die Mantisse =0 gesetzt sind. Wenn aber bei Signifikanz das zugehörige Bit =0 ist, wird eine echte Null erzeugt und es erfolgt keine Programmunterbrechung.

Eine **echte Null** ist eine Gleitpunktzahl, deren Vorzeichen, Charakteristik, Mantisse alle =0 sind. Eine echte Null kann als normales arithmetisches Ergebnis entstehen, wenn die Operanden entsprechende Werte haben, aber auch explizit erzeugt werden und zwar in folgenden Fällen:

1. Es ist Exponenten-Unterlauf entstanden und das entsprechende Maskenbit in der Programmaske ist =0.
2. Bei einer Addition oder Subtraktion ist eine Mantisse =0 entstanden und das Maskenbit für Signifikanz ist =0.
3. Der Operand eines Halbierungs-Befehls oder ein oder beide Operanden eines Multiplikations-Befehls oder der Dividend eines Divisions-Befehls haben eine Mantisse =0.

Normalisierung

Eine Größe läßt sich mit größter Genauigkeit durch eine Gleitpunktzahl darstellen, die "normalisiert" ist. Eine normalisierte Gleitpunktzahl ist eine solche, bei der die höchstwertige Mantissenstelle ungleich 0_{16} ist. Wenn die höchstwertige Sedezimalstelle der Mantisse $=0_{16}$ ist, wird die Gleitpunktzahl nicht-normalisiert genannt.

Nicht-normalisierte Gleitpunktzahlen werden normalisiert, indem die Mantisse um die Anzahl führender Sedezimal-Nullen nach links geschoben und die Charakteristik um diese Anzahl vermindert wird.

Eine Gleitpunktzahl, deren Mantisse =0 ist, kann nicht normalisiert werden; ihre Charakteristik wird entweder =0 gesetzt oder bleibt unverändert, je nachdem, ob die Gleitpunktoperation bestimmt, daß in diesem Falle eine echte Null erzeugt werden soll oder nicht.

Die Additions- und Subtraktions-Befehle mit Gleitpunktoperanden des erweiterten Formats sowie alle Multiplikations-, Divisions- und Halbierungs-Befehle normalisieren das Ergebnis automatisch. Die Addition und Subtraktion mit Gleitpunktoperanden des kurzen oder langen Formats können sowohl mit normalisiertem als auch mit nicht-normalisiertem Ergebnis veranlaßt werden. Alle anderen Befehle normalisieren ihre Ergebnisse nicht.

Bei den Befehlen, die keine Normalisierung durchführen, werden führende Sedezimal-Nullen nicht eliminiert. Das Ergebnis kann je nach Eingangsoperanden entweder normalisiert oder auch nicht-normalisiert sein.

Die Eingangsoperanden dürfen bei allen Gleitpunktbefehlen normalisiert oder nicht-normalisiert sein. Bei den Multiplikations- und Divisions-Befehlen werden die Operanden vor der eigentlichen Multiplikation bzw. Division normalisiert, bei den anderen Befehlen, die eine Normalisierung durchführen, findet diese erst bei der Herstellung des Endergebnisses statt.

Wenn bei der Bildung des Zwischenergebnisses einer Addition, Subtraktion oder Rundung die Mantisse überläuft, wird sie um eine Sedezimalstelle nach rechts geschoben; in die freigewordene Sedezimalstelle wird eine Eins (1_{16}) eingesetzt und die Charakteristik wird um Eins erhöht. Diese Schritte erfolgen auch bei den Befehlen, die im übrigen keine Normalisierung durchführen.

Gleitpunkt-Formate

Es gibt drei Formate für Gleitpunktzahlen: das "kurze", das "lange" und das "erweiterte" Format. Das kurze Format bezeichnet 32 Bit (d.h. ein "Wort") lange, das lange Format bezeichnet 64 Bit (ein "Doppelwort") lange und das erweiterte Format bezeichnet 128 Bit (zwei "Doppelworte") lange Gleitpunktzahlen.

Gleitpunktzahlen des kurzen oder langen Formats können sowohl im Hauptspeicher als auch in Gleitpunktregistern adressiert werden, Gleitpunktzahlen des erweiterten Formats jedoch nur in Gleitpunktregistern, genauer: in Gleitpunktregister-Paaren.

Kurzes Format:

V	Charakteristik	Mantisse aus 6 Sedezimalstellen
0	1	8 31

Langes Format:

V	Charakteristik	Mantisse aus 14 Sedezimalstellen
0	1	8 63

Erweitertes Format, Oberer Teil:

V	Charakteristik	höchstwertige 14 Sedezimalstellen der Mantisse aus 28 Sedezimalstellen
0	1	8 63

Erweitertes Format, Unterer Teil:

V	Charakteristik	niedrigstwertige 14 Sedezimalstellen der Mantisse aus 28 Sedezimalstellen
64	65	72 127

In allen drei Formaten bildet das Bit 0 das Vorzeichen (V). Die nächsten 7 Bit (Bit 1 bis Bit 7) stellen die Charakteristik dar. Beim kurzen und langen Format bilden die nachfolgenden 24 bzw. 56 Bit (Bit 8 bis Bit 31 bzw. Bit 8 bis Bit 63) die Mantisse, die aus 6 bzw. 14 Sedezimalstellen besteht.

Eine Gleitpunktzahl des erweiterten Formats wird durch zwei Gleitpunktzahlen des langen Formats dargestellt; diese werden "oberer Teil" und "unterer Teil" der erweiterten Gleitpunktzahl genannt.

Der obere Teil einer erweiterten Gleitpunktzahl kann eine beliebige, lange Gleitpunktzahl sein; ihr Vorzeichen und ihre Charakteristik bestimmen das Vorzeichen und die Charakteristik der gesamten Gleitpunktzahl. Ihre Mantisse bestimmt die höchstwertigen 14 Sedezimalstellen der insgesamt 28 Sedezimalstellen langen Mantisse der erweiterten Gleitpunktzahl. Wenn der obere Teil normalisiert ist, wird die gesamte Zahl als normalisiert betrachtet.

Die Mantisse des unteren Teils einer erweiterten Gleitpunktzahl bestimmt die 14 niedrigstwertigen Sedezimalstellen der insgesamt 28 Sedezimalstellen langen Mantisse der erweiterten Gleitpunktzahl. Das Vorzeichen und die Charakteristik des unteren Teils werden von den Befehlen, die erweiterte Gleitpunktzahlen verarbeiten, ignoriert; allerdings wird von den Befehlen, die erweiterte Gleitpunktzahlen erzeugen, auch im unteren Teil ein Vorzeichen und eine Charakteristik erzeugt: das Vorzeichen ist gleich dem Vorzeichen im oberen Teil, die Charakteristik ist um 14 kleiner als die des oberen Teils.

Wenn eine erweiterte Gleitpunktzahl in einem Gleitpunktregister-Paar erzeugt wird, erhält der untere Teil dasselbe Vorzeichen wie der obere Teil und seine Charakteristik wird um 14 niedriger gesetzt als die des oberen Teils, es sei denn, daß eine echte Null erzeugt wurde. Wenn durch die Subtraktion von 14 die Charakteristik des unteren Teils kleiner als Null wird, wird sie auf einen um 128 zu großen Wert gesetzt. Der Zustand "Exponenten-Unterlauf" tritt aber nur ein, wenn auch die Charakteristik des oberen Teils unterläuft.

Wenn eine erweiterte Gleitpunktzahl zu einer echten Null gemacht wird, werden sowohl der obere wie der untere Teil zu einer echten Null gemacht.

Gleitpunktregister

Es gibt 4 Gleitpunktregister mit den Nummern (Adressen) 0, 2, 4 und 6. Diese Gleitpunktregister existieren neben den Mehrzweckregistern, die bei vielen der übrigen Befehle (und bei einigen Gleitpunktbefehlen zur Basis- und Index-Adressierung) verwendet werden.

Jedes Gleitpunktregister ist 64 Bit lang. Kurze und lange Gleitpunktzahlen passen in ein einzelnes Gleitpunktregister, erweiterte Gleitpunktzahlen benötigen ein **Paar** von Gleitpunktregistern: entweder das Paar mit der Nummer 0 aus den Gleitpunktregistern 0 und 2 oder das Paar mit der Nummer 4 aus den Gleitpunktregistern 4 und 6.

Eine kurze Gleitpunktzahl belegt in einem Gleitpunktregister nur die linken 32 der insgesamt 64 Bit. Die rechten 32 Bit werden bei allen Gleitpunktbefehlen mit kurzen Operanden ignoriert bzw. bleiben unverändert, wenn kurze Gleitpunktzahlen in einem Register erzeugt werden.

Wenn das R1- oder R2-Feld eines Gleitpunktbefehls eine Registernummer enthält, die nicht gleich 0, 2, 4 oder 6 lautet oder - bei Befehlen für das erweiterte Format - eine Registerpaar-Nummer, die nicht gleich 0 oder 4 lautet, so findet eine Programmunterbrechung wegen Adreßfehlers statt.

Wertebereich von Gleitpunktzahlen

Der absolute Wertebereich W normalisierter Gleitpunktzahlen hängt von ihrem Format ab:

Kurzes Format:

$$16^{-65} \leq W \leq (1 - 16^{-6}) * 16^{63}$$

Langes Format:

$$16^{-65} \leq W \leq (1 -6^{-14}) * 16^{63}$$

Erweitertes Format:

$$16^{-65} \leq W \leq (1 - 16^{-28}) * 16^{63}$$

In allen drei Formaten ergibt sich für den (absoluten) Wertebereich W angenähert:

$$5.4 * 10^{-79} \leq W \leq 7.2 * 10^{75}$$

Schutzziffer

Das endgültige Ergebnis eines Gleitpunktbefehls hat im Falle des kurzen Formats 6, im Falle des langes Formats 14 und im Falle des erweiterten Formats 28 Sedezimalstellen; Zwischenergebnisse während der Befehlsdurchführung haben jedoch eine Sedezimalstelle mehr. Diese (niedrigstwertige) Sedezimalstelle wird **Schutzziffer** (engl. guard digit) genannt. Die Schutzziffer erhöht normalerweise die Genauigkeit des Ergebnisses. Ihre genaue Wirkung ist jedoch befehlsspezifisch und deshalb bei jedem einzelnen Gleitpunktbefehl beschrieben.

Befehlsvorrat

Es gibt 52 Gleitpunktbefehle. Diese bewirken die Addition, Subtraktion, Multiplikation und Division von zwei Gleitpunktzahlen sowie das Laden, Speichern Runden und Halbieren von einzelnen Gleitpunktzahlen. Alle Befehle verwenden entweder ein Gleitpunktregister und einen Hauptspeicher-Operanden oder zwei Gleitpunktregister.

Für kurze und lange Gleitpunktzahlen gibt es Gleitpunktbefehle für alle genannten Aufgaben, während es für erweiterte Gleitpunktzahlen nur Befehle zur Addition, Subtraktion, Multiplikation und Division gibt.

Die meisten Befehle erzeugen als Ergebnis Gleitpunktzahlen desselben Formats wie die Eingangs-Operanden.
Die Multiplikationsbefehle erzeugen jedoch aus kurzen bzw. langen Eingangsoperanden ein langes bzw. erweitertes Produkt und einige Divisionsbefehle erzeugen aus einem langen bzw. erweiterten Dividenden einen kurzen bzw. langen Quotienten. Zwei Rundungs-Befehle ermöglichen schließlich die Rundung vom erweiterten ins lange Format und vom langen ins kurze Format.

Die meisten Befehle normalisieren ihr Ergebnis. Allerdings gibt es zur Addition und zur Subtraktion Befehle, die nicht normalisieren. Manche Befehle lassen ihr Ergebnis unverändert, so daß es von den Eingangsoperanden abhängig ist, ob das Ergebnis normalisiert ist oder nicht.

Der Befehl für die erweiterte Divsion (DXR) ist nur auf Zentraleinheiten verfügbar, die über den 31-Bit-Adressierungsmodus verfügen.

Der mnemotechnischen Operationscode jedes Gleitpunktbefehls enthält an der zweiten oder dritten Stelle eine Kennzeichnung für das Format der von ihm verarbeiteten Gleitpunktzahlen. Die folgenden Buchstaben bedeuten generell:

E	kurzes Gleitpunktformat, normalisiernd
U	kurzes Gleitpunktformat, nicht-normalisierend
D	langes Gleitpunktformat, normalisierend
W	langes Gleitpunktformat, nicht-normalisierend
X	erweitertes Gleitpunktformat, normalisierend

Programmierhinweise

– Eine lange Gleitpunktzahl kann in eine erweiterte Gleitpunktzahl verwandelt werden, indem man ihr eine lange Gleitpunktzahl anfügt, deren Mantisse =0 ist; dies kann insbesondere eine echte Null sein. Der umgekehrte Vorgang, nämlich die Konvertierung einer erweiterten in eine lange Gleitpunktzahl geschieht entweder durch den LRDR-Befehl oder einfach durch das Weglassen des unteren Teils.

– Wenn kein Fall von Exponenten-Überlauf oder -Unterlauf vorliegt, dann stellt die zweite lange Gleitpunktzahl einer erweiterten Gleitpunktzahl genau dann deren korrekten unteren Teil dar, wenn ihre Charakteristik um mindestens 14 kleiner ist als die der ersten langen Gleitpunktzahl. Wenn die Differenz der Charakteristiken der beiden langen Gleitpunktzahlen weniger als 14 beträgt und die erweiterte Gleitpunktzahl ist keine echte Null, dann ist der untere Teil inkorrekt.

– Bis zu drei führende Bit einer normalisierten Gleitpunktzahl können = 0 sein, weil die Normalisierung sedezimalziffernweise, also 4 bitweise erfolgt.

– Durch das BS2000 werden alle vier Maskenbit der Programmaske mit 1 vorbesetzt. Deshalb erfolgt unter den oben beschriebenen Bedingungen für Exponenten-Unterlauf und für Signifikanz normalerweise eine Programmunterbrechung. Mit dem Befehl SPM (Setzen Programmaske) kann das Anwenderprogramm jedoch die Vorbesetzung ändern.

– Bei der Normalisierung einer erweiterten Gleitpunktzahl wird die gesamte 28-stellige Mantisse verwendet. Der untere Teil braucht für sich nicht normalisiert zu sein, obwohl es die erweiterte Gleitpunktzahl ist.

- Zur Konvertierung einer 32 Bit langen Festpunktzahl in eine lange Gleitpunktzahl und umgekehrt einer langen Gleitpunktzahl in eine 32 Bit lange Festpunktzahl sind folgende Befehlsfolgen möglich:

```
Name         Operation    Operanden
             .
FPTOFL       EQU          *                     Festpunktzahl aus MZ-Reg 0
             ST           0,TMPDWORD+4
             XI           TMPDWORD+4,X'80'
             LD           0,TMPDWORD
             LE           0,TWOEX31
             SD           0,TWOEX31             Gleitpunktzahl in GP-Reg 0

FLTOFP       EQU          *                     Gleitpunktzahl aus GP-Reg 0
             AW           0,TWOEX31
             BM           TOOSMALL              Fehler: <-2^31
             CE           0,TWOEX31
             BNE          TOOBIG                Fehler: ≥+2^31
             STD          0,TMPDWORD
             XI           TMPDWORD+4,X'80'
             L            0,TMPDWORD+4          Festpunktzahl in MZ-Reg 0
             .

* Daten dazu:

TMPDWORD     DS           D
TWOEX31      DC           X'4E00000080000000'   8*16^-7*16^14 =2^31
             .
```

Die FPTOFL-Routine tranformiert zunächst die zu konvertierende Festpunktzahl aus dem Bereich $-2^{31}...+2^{31}-1$ in den Bereich $0...2^{32}-1$, indem sie -modulo 2^{32}- den Wert 2^{31} addiert; dies geschieht durch Invertierung der Vorzeichenstelle mittels XI. Danach macht sie diese Zahl zum rechten Teil der Mantisse einer langen Gleitpunktzahl mit dem Exponenten 14, d.h. der Charakteristik $(64+14)_{10} = (4E)_{16}$. Schließlich subtrahiert sie davon die zunächst addierte 2^{31} wieder und normalisiert das Ergebnis.

Die FLTOFP-Routine addiert zunächst nicht-normalisiert den Wert 2^{31} auf die zu konvertierende Gleitpunktzahl und schiebt dabei etwaige nicht-ganzzahlige Sedezimalstellen rechts hinaus. Es erfolgt keine Rundung. Wenn die Summe <0 ist, war die Gleitpunktzahl $<-2^{31}$, also zu klein für den Wertebereich von Festpunktzahlen, wenn die Summe $\geq 2^{32}$ ist, war sie zu groß; letzteres wird daran erkannt, daß die linken 6 Sedezimalstellen $\neq 0$ sind. Vom rechten Teil der Mantisse muß noch die zunächst addierte 2^{31} subtrahiert werden, was wie bei FPTOFL durch einen Befehl XI geschieht. Der abschließende L-Befehl lädt die fertige Festpunktzahl ins Mehrzweckregister 0.

Add Normalized

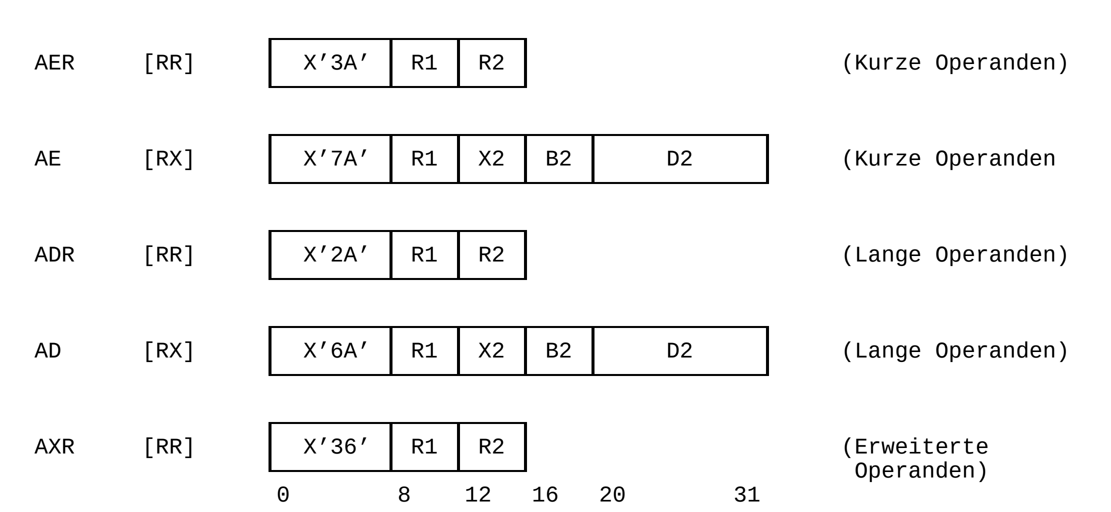
Funktion

Die Befehle AER, AE, ADR, AD und AXR addieren zwei Gleitpunktzahlen. Die normalisierte Summe ersetzt den ersten Operanden.
Die Anzeige wird gemäß dem Wert der Summe gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Summanden, kurze Summe:			
	AER	R1,R2	R1,R2 =0, 2, 4 oder 6
	AE	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* lange Summanden, lange Summe:			
	ADR	R1,R2	R1,R2 =0, 2, 4 oder 6
	AD	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* erweiterte Summanden, erweiterte Summe:			
	AXR	R1,R2	R1,R2 =0 oder 4

Maschinenformate



Beschreibung

Zunächst werden die Charakteristiken der beiden Operanden verglichen; die Mantisse des Operanden mit der kleineren Charakteristik wird um die Differenz der Charakteristiken nach rechts geschoben und seine Charakteristik wird um denselben Betrag erhöht, so daß die Charakteristiken gleich werden. Die zuletzt hinausgeschobene Sedezimalziffer wird als Schutzziffer aufbewahrt. Die Schutzziffer des anderen Operanden, - oder beider Operanden, wenn vor der Addition die Charakteristiken gleich waren - wird =0 gesetzt.

Anschließend werden die beiden Mantissen einschließlich der Schutzziffern unter Berücksichtigung der Vorzeichen addiert. Die Summe bildet ein Zwischenergebnis. Das Zwischenergebnis besteht beim kurzen Format aus 7, beim langen Format aus 15 und beim erweiterten Format aus 29 Sedezimalstellen.

Wenn ein Überlauf aufgetreten ist, wird das Zwischenergebnis um eine Sedezimalstelle nach rechts geschoben, dann in die links freigewordene Sedezimalstelle eine 1_{16} eingesetzt und die Charakteristik um 1 erhöht.

Signifikanz tritt ein, wenn das Zwischenergebnis einschließlich Schutzziffer =0 ist. Wenn in diesem Fall das Bit für Signifikanz in der Programmaske =1 ist (BS2000-Standard), erfolgt eine Programmunterbrechung, andernfalls erfolgt keine Programmunterbrechung und als Endergebnis wird eine echte Null erzeugt.

Wenn das Zwischenergebnis einschließlich Schutzziffer ungleich 0 ist, wird es normalisiert, d.h. solange nach links geschoben, bis die höchstwertige Sedezimalstelle von 0_{16} verschieden ist. Rechts freiwerdende Sedezimalstellen werden mit 0_{16} gefüllt. Die Charakteristik wird um die Anzahl der geschobenen Sedezimalstellen vermindert.

Zuletzt wird das normalisierte Zwischenergebnis auf 6 bzw. 14 bzw. 28 Sedezimalstellen gekürzt und zusammen mit der zuvor ermittelten Charakteristik zum Endergebnis gemacht. Beim erweiterten Format wird im unteren Teil der Gleitpunkt-Summe eine Charakteristik erzeugt, die um 14 kleiner ist als die des oberen Teils und das Vorzeichen des unteren Teils wird gleich dem des oberen Teils gesetzt.

Exponenten-Überlauf tritt ein, wenn die Charakteristik des Endergebnisses größer als 127 wird. Es folgt dann eine Programmunterbrechung: Vorzeichen und Mantisse sind korrekt, aber die Ergebnis-Charakteristik(en) sind um 128 zu klein.
Exponenten-Unterlauf tritt ein, wenn die Charakteristik des Endergebnisses kleiner als 0 wird. Wenn in diesem Falle das Bit für Exponenten-Unterlauf in der Programmaske =1 ist (BS2000-Standard), erfolgt eine Programmunterbrechung: Vorzeichen und Mantisse sind korrekt, aber die Ergebnis-Charakteristik(en) sind um 128 zu groß; andernfalls erfolgt keine Programmunterbrechung und als Endergebnis wird eine echte Null erzeugt.
Beim Befehl AXR tritt Exponenten-Unterlauf nicht ein, wenn nur beim unteren Teil des Endergebnisses die Charakteristik kleiner als 0 wird. In diesem Falle ist dessen Charakteristik um 128 zu groß gesetzt.

Anzeige

0~Zero	Die Mantisse des Ergebnisses ist = 0; das Vorzeichen ist positiv.
1~Minus	Ergebnis ist < 0.
2~Plus	Ergebnis ist > 0.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	AE, AD: Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	falsche Gleitpunktregisterangabe oder D2(X2,B2) keine (Doppel-)Wortgrenze
Exponenten-Überlauf	X’64’	Summen-Charakteristik > 127
Signifikanz	X’6C’	Mantisse =0, Charakteristik 0 und Maskenbit für Signifikanz =1
Exponenten-Unterlauf	X’70’	Summen-Charakteristik < 0

Programmierhinweise

- Die Vertauschung der beiden Operanden einer normalisierenden Addition ändert in keinem Falle das Resultat.
- Die normalisierende Addition normalisiert die Summe, aber nicht die Summanden.
- Durch das BS2000 werden die Bit für Exponenten-Unterlauf und Signifikanz in der Programmaske mit 1 vorbesetzt, so daß in den oben genannten Fällen eine Programmunterbrechung eintritt. Mit dem Befehl SPM (Setzen Programmaske) kann ein Anwenderprogramm die Vorbesetzung ändern.
- Bei AE und AER werden die rechten 32 Bit der verwendeten Gleitpunktregister ignoriert bzw. bleiben unverändert.
- R2 darf =R1 sein.

Beispiel

Name	Operation	Operanden
	.	
	DS	0D
FLNO1	DC	X’3F11111111111111’
FLNO2	DC	X’C001111111111111’
	.	
	.	
	.	
	LD	2,FLNO1
	AD	2,FLNO2
	.	

Das Endergebnis im Gleitpunktregister 2 lautet X’3210000000000000’ zusammen mit der Anzeige 2~Plus. Nach dem Charakteristik-Abgleich hatte der erste Operand den Wert X’4001111111111111’ und die Schutzziffer $=1_{16}$. Das Zwischenergebnis lautete X’4000000000000001’.

Add Unnormalized

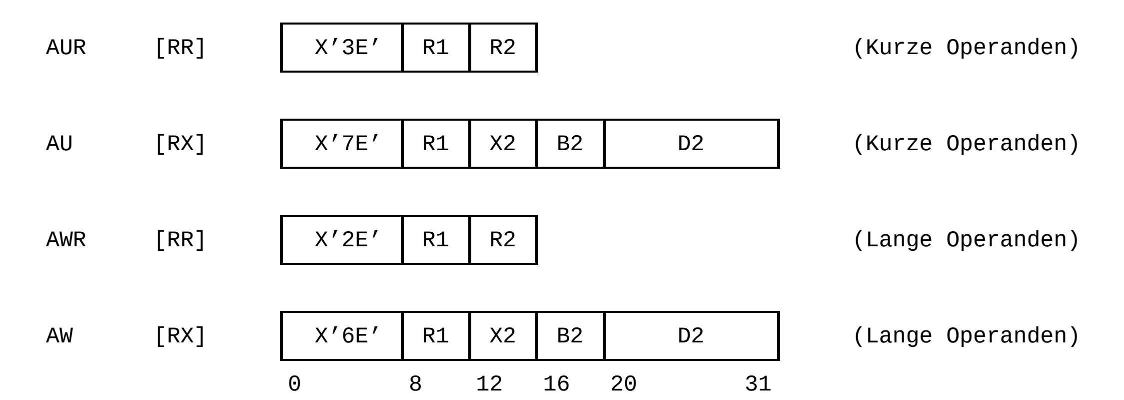
Funktion

Die Befehle der AUR, AU, AWR und AW addieren zwei Gleitpunktzahlen. Die Summe ersetzt den ersten Operanden; sie wird nicht normalisiert.
Die Anzeige wird gemäß dem Wert der Summe gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Summanden, kurze Summe:			
	AUR	R1,R2	R1,R2 =0, 2, 4 oder 6
	AU	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* lange Summanden, lange Summe:			
	AWR	R1,R2	R1,R2 =0, 2, 4 oder 6
	AW	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und

Maschinenformate



Beschreibung

Zunächst werden die Charakteristiken der beiden Operanden verglichen; die Mantisse des Operanden mit der kleineren Charakteristik wird um die Differenz der Charakteristiken nach rechts geschoben und seine Charakteristik wird um denselben Betrag erhöht, so daß die Charakteristiken gleich werden. Die zuletzt hinausgeschobene Sedezimalziffer wird als Schutzziffer aufbewahrt. Die Schutzziffer des anderen Operanden - oder beider Operanden, wenn vor der Addition die Charakteristiken gleich waren - wird $=0_{16}$ gesetzt.
Anschließend werden die beiden Mantissen einschließlich der Schutzziffern unter Berücksichtigung der Vorzeichen addiert. Die Summe bildet ein Zwischenergebnis. Das Zwischenergebnis besteht beim kurzen Format aus 7 und beim langen Format aus 15 Sedezimalstellen.

Wenn ein Überlauf aufgetreten ist, wird das Zwischenergebnis um eine Sedezimalstelle nach rechts geschoben, dann in die links freigewordene Sedezimalstelle eine 1_{16} eingesetzt und die Charakteristik um 1 erhöht.

Signifikanz tritt ein, wenn das Zwischenergebnis **ausschließlich Schutzziffer** =0 ist. Wenn in diesem Fall das Bit für Signifikanz in der Programmaske =1 ist (BS2000-Standard), erfolgt eine Programmunterbrechung, andernfalls erfolgt keine Programmunterbrechung und als Endergebnis wird eine echte Null erzeugt.
Das Zwischenergebnis wird - ohne vorherige Normalisierung - auf 6 bzw. 14 Sedezimalstellen gekürzt und zusammen mit der zuvor ermittelten Charakteristik zum Endergebnis gemacht.

Exponenten-Überlauf tritt ein, wenn die Charakteristik des Endergebnisses größer als 127 wird. Es erfolgt dann eine Programmunterbrechung: Vorzeichen und Mantisse sind korrekt, aber die Ergebnis-Charakteristik(en) sind um 128 zu klein.

Exponenten-Unterlauf kann nicht eintreten.

Anzeige

0~Zero	Die Mantisse des Ergebnisses ist = 0; das Vorzeichen ist positiv.
1~Minus	Das Ergebnis ist < 0.
2~Plus	Das Ergebnis ist > 0.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	AU, AW: Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	falsche Gleitpunktregisterangabe oder D2(X2,B2) keine (Doppel-)Wortgrenze
Exponenten-Überlauf	X’64’	Summen-Charakteristik > 127
Signifikanz	X’6C’	Mantisse =0, Charakteristik 0 und Maskenbit für Signifikanz =1

Programmierhinweise

- Die Vertauschung der beiden Operanden einer nicht-normalisierenden Addition ändert in keinem Falle das Resultat.
- Durch das BS2000 wird das Bit für Signifikanz in der Programmaske mit 1 vorbesetzt, so daß im oben genannten Fall eine Programmunterbrechung eintritt. Mit dem Befehl SPM (Setzen Programmaske) kann ein Anwenderprogramm die Vorbesetzung ändern.
- Bei AU und AUR werden die rechten 32 Bit der verwendeten Gleitpunktregister ignoriert und bleiben unverändert.
- Die nicht-normalisierende Addition ist bis auf folgende Unterschiede gleichwertig zur normalisierenden Addition:
  - Das Ergebnis wird nicht normalisiert.
  - Exponenten-Unterlauf kann nicht eintreten.
  - Die Schutzziffer wird nicht zur Ermittlung von Signifikanz herangezogen.
- Für erweiterte Gleitpunktoperanden gibt es zwar einen Befehl zur normalisierenden Addition (AXR), aber keinen Befehl zur nicht-normalisierenden Addition.

Beispiel

Siehe das Beispiel für nicht-normalisierende Subtraktion (SU).

Compare

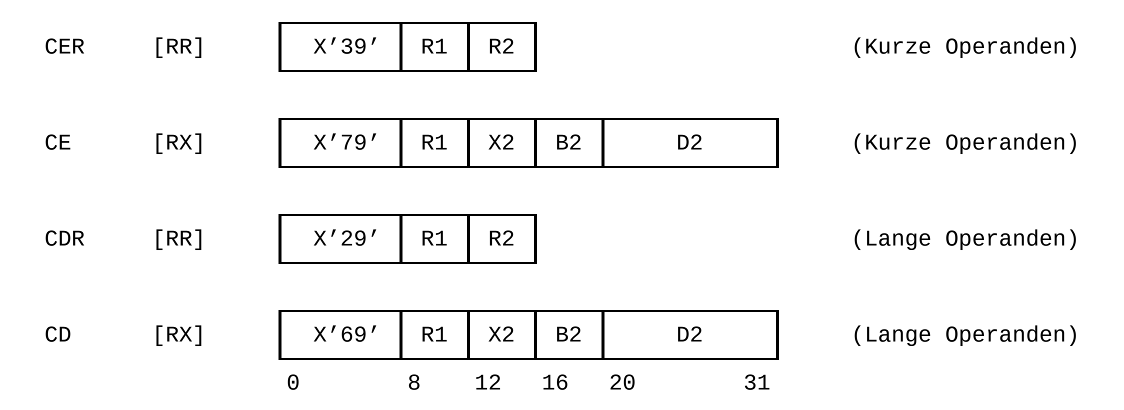
Funktion

Die Befehle CER, CE, CDR und CD vergleichen zwei Gleitpunktzahlen.
Die Anzeige wird gemäß dem Vergleichsergebnis gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Operanden:			
	CER	R1,R2	R1,R2 =0, 2, 4 oder 6
	CE	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* lange Operanden:			
	CDR	R1,R2	R1,R2 =0, 2, 4 oder 6
	CD	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6

Maschinenformate



Beschreibung

Der Vergleich wird so durchgeführt, als würde eine normalisierende Subtraktion stattfinden, bei der die Differenz nicht abgespeichert wird. Die Anzeige wird auf 0~Equal gesetzt, wenn beide Operanden einschließlich Schutzziffer gleich sind und auf 1~Low bzw. 2~High gesetzt, wenn der erste Operand kleiner bzw. größer als der zweite ist.

Exponenten-Unterlauf, Exponenten-Überlauf und Signifikanz können nicht eintreten.

Anzeige

0~Equal	Der Operand1 ist einschließlich Schutzziffer = Operand2.
1~Low	Der Operand1 ist < Operand2.
2~High	Der Operand1 ist > Operand2.
3~Overflow	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	CE, CD: Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	falsche Gleitpunktregisterangabe oder D2(X2,B2) keine (Doppel-)Wortgrenze

Programmierhinweise

- Zwei Operanden, die beide die Mantisse 0 haben, erbringen die Anzeige 0~Equal, auch wenn sie unterschiedliche Vorzeichen oder Charakteristiken haben.
- Es ist noch keine hinreichende Bedingung für Ungleichheit, wenn die Charakteristiken der zwei Operanden verschieden sind.
- Die Befehle CE und CER vergleichen nur die linken 32 Bit ihrer Operanden; deshalb ist es möglich, daß sie Gleichheit anzeigen, wo CD und CDR dies nicht tun.
- Es gibt keinen Befehl zum Vergleich von zwei Gleitpunktoperanden des erweiterten Formats.

Beispiel

Name	Operation	Operanden
	.	
	DS	0F
FLNO1	DC	X’48001000’ $=16^5+0$
FLNO2	DC	X’47010001’ $=16^5+16$
	.	
	.	
	.	
	LE	6,FLNO1
	CE	6,FLNO2 ergibt Anzeige 1~Low
	.	

Mit diesen Daten wird durch die obigen Befehle die Anzeige auf 1~Low gesetzt, weil nach dem Charakteristikabgleich die Schutzziffern ungleich sind. Dagegen würde die Anzeige bereits auf 0~Equal gesetzt, wenn FLNO2 nur um Eins kleiner wäre, also den Wert X’4610000F’ $=16^5+15$ hätte, da dann die beiden Schutzziffern gleich sind.

Divide

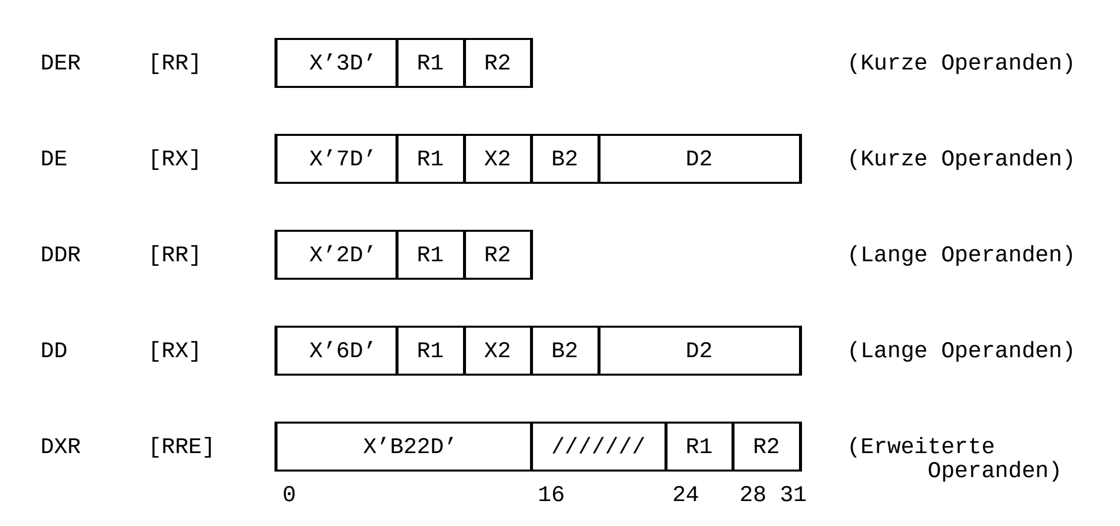
Funktion

Die Befehle DER, DE, DDR, DD und DXR dividieren zwei Gleitpunktzahlen. Der normalisierte Quotient ersetzt den ersten Operanden. Der Befehl DXR ist nur im Befehlsvorrat der Zentraleinheiten verfügbar, die über den 31-Bit-Adressierungsmodus verfügen.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Operanden, kurzer Quotient:			
	DER	R1,R2	R1,R2 =0, 2, 4 oder 6
	DE	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* lange Operanden, langer Quotient:			
	DDR	R1,R2	R1,R2 =0, 2, 4 oder 6
	DD	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* erweiterte Operanden, erweiterter Quotient:			
	DXR	R1,R2	R1,R2 =0 oder 4

Maschinenformate



Beschreibung

Der Gleitpunkt-Operand1 ist der Dividend, der Gleitpunkt-Operand2 ist der Divisor. Der normalisierte Quotient ersetzt den ersten Operanden. Es wird kein Divisionsrest erzeugt.

Zunächst werden Dividend und Divisor normalisiert, so daß sie keine führende Sedezimal-Nullen haben und es werden ihre Charakteristiken entsprechend angepaßt (vermindert). Die Normalisierung geschieht intern, die Eingangs-Operanden werden nicht verändert; wenn der Dividend - dem Betrage nach - größer ist als der Divisor, wird der Dividend um eine Sedezimalstelle nach rechts geschoben und seine Charakteristik um 1 erhöht.

Die Charakteristiken der beiden Operanden werden subtrahiert und die beiden (normalisierten) Mantissen werden dividiert. Der resultierende Quotient bildet zusammen mit der Charakteristikdifferenz plus 64 und dem algebraisch ermittelten Vorzeichen ein Zwischenergebnis. Bei der Division werden alle Sedezimalziffern beider Mantissen berücksichtigt.

Das Zwischenergebnis wird zuletzt bei DER und DE auf 6 Sedezimalstellen, bei DDR und DD auf 14 Sedezimalstellen und bei DXR auf 28 Sedezimalstellen gekürzt und zum Endergebnis gemacht. Dieses ist stets normalisiert.

Exponenten-Überlauf tritt ein, wenn die Charakteristik des Endergebnisses größer als 127 wird und seine Mantisse von Null verschieden ist. Es erfolgt dann eine Programmunterbrechung: Mantisse und Vorzeichen sind korrekt, aber die Charakteristik des Endergebnisses ist um 128 zu klein. Bei DXR ist ggf. auch die Charakteristik des unteren Teils um 128 zu klein.
Exponenten-Unterlauf tritt ein, wenn die Charakteristik des Endergebnisses kleiner als 0 wird und seine Mantisse von Null verschieden ist. Wenn in diesem Falle das Bit für Exponenten-Unterlauf in der Programmaske =1 ist (BS2000-Standard), erfolgt eine Programmunterbrechung: Mantisse und Vorzeichen sind korrekt, aber die Charakteristik ist um 128 zu groß; andernfalls erfolgt keine Programmunterbrechung und als Quotient wird eine echte Null erzeugt. Bei DXR tritt Exponenten-Unterlauf nicht ein, wenn nur beim unteren Teil des Endergebnisses die Charakteristik kleiner als 0 wird. In diesem Falle ist dessen Charakteristik um 128 zu groß.

Exponenten-Überlauf oder -Unterlauf kann erst beim Endergebnis auftreten, nicht schon, wenn während der Zwischenrechnungen eine Charakteristik über- oder unterläuft.

Divisionsfehler tritt ein, wenn die Mantisse des Divisors =0 ist (auch wenn der Dividend ebenfalls =0 ist). Es erfolgt eine Programmunterbrechung.

Wenn die Mantisse des Dividenden =0 ist, aber die des Divisors ≠0 ist, wird als Endergebnis eine echte Null erzeugt.

Das Vorzeichen des Quotienten wird nach den üblichen algebraischen Regeln ermittelt; eine echte Null hat jedoch stets ein positives Vorzeichen.

Die Bitstellen 16 bis 23 des Befehls DXR werden ignoriert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	DE, DD: Lesezugriff auf Operand2 unmöglich
Falscher Operations-Code	X’58’	DXR auf einer Zentraleinheit versucht, die nicht 31-Bit fähig ist
Adreßfehler	X’5C’	falsche Gleitpunktregisterangabe oder D2(X2,B2) keine (Doppel-)Wortgrenze
Exponenten-Überlauf	X’64’	Quotient-Charakteristik > 127
Divisionsfehler	X’68’	Divisor-Mantisse =0
Exponenten-Unterlauf	X’70’	Quotient-Charakteristik < 0

Programmierhinweise

- Bei DER und DE werden die rechten 32 Bit des Gleitpunktregisters R1 für die Mantissen-Division ignoriert und verbleiben unverändert, ebenso bei DER die rechten 32 Bit des Gleitpunktregisters R2.
- Durch das BS2000 wird das Bit für Exponenten-Unterlauf in der Programmaske mit 1 vorbesetzt, so daß unter den oben beschriebenen Bedingungen standardmäßig eine Programmunterbrechung erfolgt. Mit dem Befehl SPM kann das Anwenderprogramm jedoch die Vorbesetzung ändern.
- R2 darf =R1 sein.

Beispiel

Name	Operation	Operanden
	.	
	DS	0D
DIVIDEND	DC	X’00100000’ $16^{-64}*(16^{-1}+8*16^{-7}) = 1*16^{-71}*(16^{6}+8)$
	DC	X’80000000’
DIVISOR	DC	X’80020000’ $-16^{-64}*(2*16^{-2}+16^{-7}) =-8^{-1}*16^{-71}*(16^{6}+8)$
	DC	X’10000000’
	.	
	.	
	LD	6,DIVIDEND
	DD	6,DIVISOR Ergebnis im Gleitpunktregister 6:
	.	X’C180000000000000’ $= -16^{+1}*8*16^{-1} = -8$

Man beachte, daß durch die Normalisierung des Divisors "eigentlich" Exponenten-Unterlauf entsteht; da dieser jedoch nur beim Zwischenergebnis, nicht beim Endergebnis auftritt, erfolgt keine Programmunterbrechung.

Halve

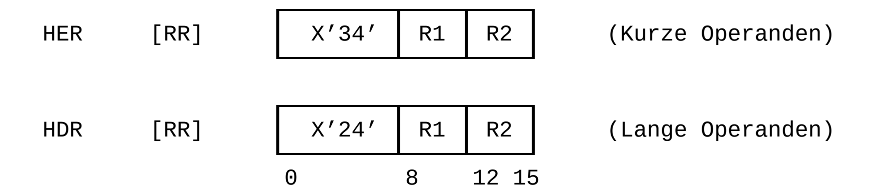
Funktion

Die Befehle HER und HDR dividieren die Gleitpunktzahl im Gleitpunktregister R2 durch +2 und speichern das normalisierte Ergebnis ins Gleitpunktregister R1.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Operanden:			
	HER	R1,R2	R1,R2 =0, 2, 4 oder 6
* lange Operanden:			
	HDR	R1,R2	R1,R2 =0, 2, 4 oder 6

Maschinenformate



Beschreibung

Die bei HER 6-stellige und bei HDR 14-stellige Mantisse der Gleitpunktzahl im Gleitpunktregister R2 wird um ein Bit nach rechts geschoben und die freigewordene Bitstelle mit 0 gefüllt. Die rechts aus der Mantisse hinausgeschobene Bitstelle wird links in die Schutzziffer plaziert und die übrigen drei Bit der Schutzziffer werden =0 gesetzt.

Das so entstandene Zwischenergebnis wird einschließlich der Schutzziffer normalisiert und das Endergebnis in das Gleitpunktregister R1 gespeichert.

Exponenten-Unterlauf tritt ein, wenn die Charakteristik des Endergebnisses kleiner als 0 wird und seine Mantisse von Null verschieden ist. Wenn in diesem Falle das Bit für Exponenten-Unterlauf in der Programmaske =1 ist (BS2000-Standard), erfolgt eine Programmunterbrechung:

Mantisse und Vorzeichen sind korrekt, aber die Charakteristik ist um 128 zu groß; andernfalls erfolgt keine Programmunterbrechung und als Endergebnis wird eine echte Null erzeugt.

Wenn die Mantisse des Eingangsoperanden (im Gleitpunktregister R2) =0 ist, wird als Endergebnis eine echte Null erzeugt. Signifikanz oder Exponenten-Unterlauf tritt in diesem Fall nicht auf.

Das Vorzeichen des Ergebnisses ist gleich dem des Eingangs-Operanden; eine echte Null hat jedoch stets ein positives Vorzeichen.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßfehler Exponenten-Unterlauf	X’5C’ X’70’	falsche Gleitpunktregisterangabe Ergebnis-Charakteristik < 0

Programmierhinweise

– Bei HER werden die rechten 32 Bit des Gleitpunktregisters R2 ignoriert und und die rechten 32 Bit des Gleitpunktregisters R1 nicht verändert.

– Das Ergebnis eines HER- oder HDR-Befehls ist in allen Fällen identisch mit dem Ergebnis einer Gleitpunktdivision mittels DER bzw. DDR und einem Divisor = 2.

– Durch das BS2000 wird das Maskenbit fur Exponenten-Unterlauf in der Programmaske mit 1 vorbesetzt, so daß unter den oben beschriebenen Bedingungen eine Programmunterbrechung erfolgt. Mit dem Befehl SPM kann das Anwenderprogramm jedoch die Vorbesetzung ändern.

– Eine echte Null kann nur entstehen, wenn der Eingangsoperand eine Mantisse =0 hat oder wenn Exponenten-Unterlauf auftritt und das Bit für Exponenten-Unterlauf in der Programmaske =0 ist.

– R2 darf =R1 sein.

Beispiel

Name	Operation	Operanden
	.	
	LE	6,FLNO
	HER	4,6
	.	
	.	
	.	
	DS	0F
FLNO	DC	X’86000001’ $-1*16^{-6}*16^{-58}$
	.	

Im Gleitpunktregister 4 steht bei Befehlsabschluß der Wert:
X’80800000’ = $-0{,}5*16^{-64}$.

Die Anzeige und der rechte Teil des Gleitpunktregisters 4 sind unverändert.

Load Complement

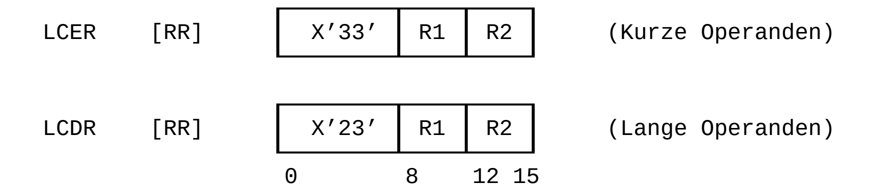
Funktion

Die Befehle LCER und LCDR laden die Gleitpunktzahl im Gleitpunktregister R2 mit umgekehrtem Vorzeichen in das Gleitpunktregister R1 und setzen die Anzeige gemäß dem Wert in R1.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Operanden:			
	LCER	R1,R2	R1,R2 =0, 2, 4 oder 6
* lange Operanden:			
	LCDR	R1,R2	R1,R2 =0, 2, 4 oder 6

Maschinenformate



Beschreibung

Die kurze (LCER) bzw. lange (LCDR) Gleitpunktzahl im Gleitpunktregister R2 wird mit umgekehrtem Vorzeichen in das Gleitpunktregister R1 übertragen. Es findet keine Normalisierung statt.

Das Vorzeichen wird auch umgekehrt, wenn die Mantisse des Eingangs-Operanden =0 ist; die Anzeige ist jedoch in diesem Fall auf 0~Zero gesetzt.

Anzeige

0~Zero	Mantisse des Ergebnisses ist = 0.
1~Minus	Ergebnis ist < 0.
2~Plus	Ergebnis ist > 0.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßfehler	X’5C’	falsche Gleitpunktregisterangabe

Programmierhinweise

- R1 darf =R2 sein
- Der Befehl LCER überträgt nur die linken 32 Bit des Gleitpunktregisters R2 und läßt die rechten 32 Bit des Gleitpunkregisters R1 unverändert.

Load

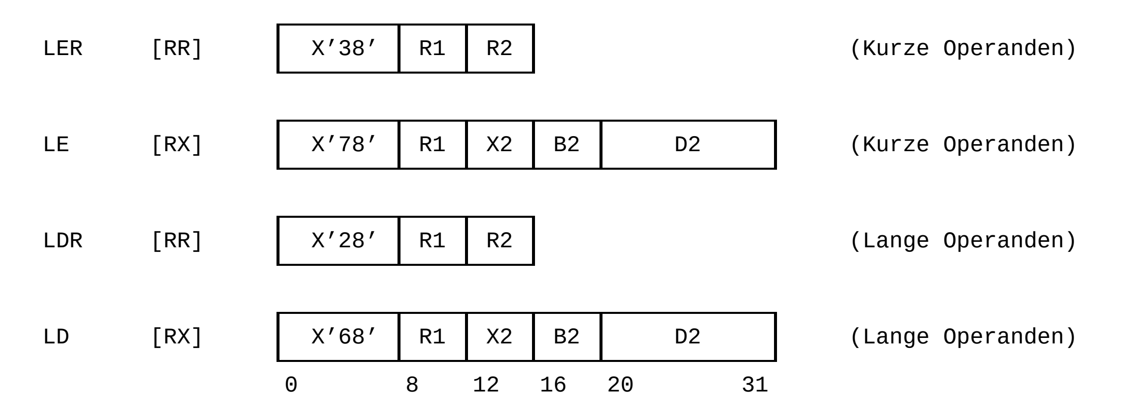
Funktion

Die Befehle LER, LE, LDR und LD laden eine Gleitpunktzahl in ein Gleitpunktregister. Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Operanden:			
	LER	R1,R2	R1,R2 =0, 2, 4 oder 6
	LE	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* lange Operanden:			
	LDR	R1,R2	R1,R2 =0, 2, 4 oder 6
	LD	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und

Maschinenformate



Beschreibung

Die kurze (LE und LER) bzw. lange (LD und LDR) Gleitpunktzahl in dem Gleitpunktregister R2 (LER und LDR) bzw. in dem Hauptspeicherwort (LE) oder Hauptspeicher-Doppelwort (LD) wird in das Gleitpunktregister R1 geladen. Es findet keine Normalisierung statt.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Adreßfehler	X’48’ X’5C’	LE, LD: Lesezugriff auf Operand2 unmöglich falsche Gleitpunktregisterangabe oder D2(X2,B2) keine (Doppel-)Wortgrenze

Programmierhinweise

Die Befehle LE und LER lassen die rechten 32 Bit des Gleitpunkregisters R1 unverändert.

Load Negative

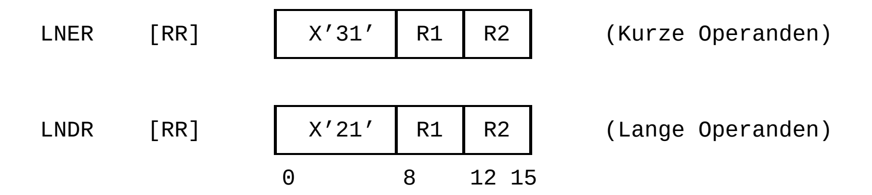
Funktion

Die Befehle LNER und LNDR laden die Gleitpunktzahl im Gleitpunktregister R2 mit negativem Vorzeichen in das Gleitpunktregister R1 und setzen die Anzeige gemäß dem Wert in R1.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Operanden:			
	LNER	R1,R2	R1,R2 =0, 2, 4 oder 6
* lange Operanden:			
	LNDR	R1,R2	R1,R2 =0, 2, 4 oder 6

Maschinenformate



Beschreibung

Die kurze (LNER) bzw. lange (LNDR) Gleitpunktzahl im Gleitpunktregister R2 wird mit negativem Vorzeichen in das Gleitpunktregister R1 übertragen. Es findet keine Normalisierung statt.

Das Vorzeichen wird auch negativ gesetzt, wenn die Mantisse des Eingangs-Operanden =0 ist; die Anzeige wird jedoch in diesem Fall auf 0~Zero gesetzt.

Anzeige

0~Zero	Mantisse des Ergebnisses ist = 0.
1~Minus	Ergebnis ist < 0.
2	Nicht verwendet.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßfehler	X’5C’	falsche Gleitpunktregisterangabe

Programmierhinweise

- R1 darf =R2 sein
- Der Befehl LNER überträgt nur die linken 32 Bit des Gleitpunktregisters R2 und läßt die rechten 32 Bit des Gleitpunkregisters R1 unverändert.

Load Positive

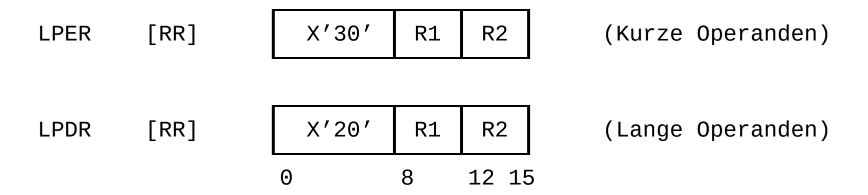
Funktion

Die Befehle LPER und LPDR laden die Gleitpunktzahl im Gleitpunktregister R2 mit positivem Vorzeichen in das Gleitpunktregister R1 und setzen die Anzeige gemäß dem Wert in R1.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Operanden:			
	LPER	R1,R2	R1,R2 =0, 2, 4 oder 6
* lange Operanden:			
	LPDR	R1,R2	R1,R2 =0, 2, 4 oder 6

Maschinenformate



Beschreibung

Die kurze (LPER) bzw. lange (LPDR) Gleitpunktzahl im Gleitpunktregister R2 wird mit positivem Vorzeichen in das Gleitpunktregister R1 übertragen. Es findet keine Normalisierung statt.

Das Vorzeichen wird auch positiv gesetzt, wenn die Mantisse des Eingangs-Operanden =0 ist; die Anzeige wird jedoch in diesem Fall auf 0~Zero gesetzt.

Anzeige

0~Zero	Mantisse des Ergebnisses ist = 0.
1	Nicht verwendet.
2~Plus	Ergebnis ist > 0.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßfehler	X’5C’	falsche Gleitpunktregisterangabe

Programmierhinweise

- R1 darf =R2 sein
- Der Befehl LPER überträgt nur die linken 32 Bit des Gleitpunktregisters R2 und läßt die rechten 32 Bit des Gleitpunkregisters R1 unverändert.

Load Rounded

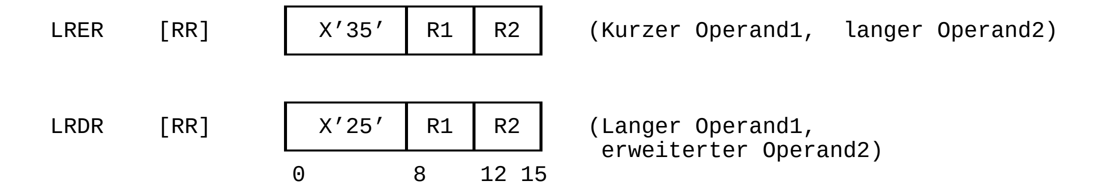
Funktion

Die Befehle LRER und LRDR laden die Gleitpunktzahl im Gleitpunktregister(-Paar) R2 in das Gleitpunktregister R1 und runden sie dabei auf das nächstkleinere Gleitpunktformat. Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurzer Operand1, langer Operand2:			
	LRER	R1,R2	R1,R2 =0, 2, 4 oder 6
* langer Operand1, erweiterter Operand2:			
	LRDR	R1,R2	R1 =0, 2, 4 oder 6 und R2 =0 oder 4

Maschinenformate



Beschreibung

Die lange (LRER) bzw. erweiterte (LRDR) Gleitpunktzahl im Gleitpunktregister R2 bzw. im Gleitpunkregister-Paar R2 und R2+2 wird auf das kurze (LRER) bzw. lange (LRDR) Gleitpunktformat gerundet und in das Gleitpunktregister R1 übertragen. Es findet keine Normalisierung statt. Das Vorzeichen bleibt unverändert.

Die Rundung besteht darin, daß zur Bitstelle 32 bzw. 72 der Mantisse des zweiten Operanden eine Eins addiert wird und ein etwaiger Übertrag auf die höherwertigen Mantissenstellen weitergereicht wird.

Wenn bei der Rundung ein Übertrag über die höchstwertige Sedezimalstelle der Mantisse auftritt, wird diese um eine Sedezimalstelle nach rechts geschoben, in die freigewordene Stelle eine 1_{16} eingesetzt und die Charakteristik um Eins erhöht.

Exponenten-Überlauf tritt ein, wenn die Ergebnis-Charakteristik größer als 127 wird. Es findet dann eine Programmunterbrechung statt, wobei Mantisse und Vorzeichen des Ergebnisses korrekt sind, aber die Charakteristik um 128 zu klein ist.

Exponenten-Unterlauf und Signifikanz können nicht eintreten.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßfehler	X'5C'	falsche Gleitpunktregisterangabe
Exponenten-Überlauf	X'64'	Ergebnis-Charakteristik > 127

Programmierhinweise

- R1 darf =R2 sein. Man beachte jedoch, daß bei LRER der rechte Teil des Gleitpunktregisters R1 bzw. bei LRDR der Inhalt des Gleitpunktregisters R1+2 nach der Operation nicht mehr zur Interpretation des Ergebnisses herangezogen werden darf (siehe Beispiel).
- Der Befehl LRER läßt die rechten 32 Bit des Gleitpunkregisters R1 unverändert.

Beispiel

Name	Operation	Operanden
	.	
	LD	0,=XL8'C6FFFFFF890ABCDE'
	LRER	0,0
	CD	0,=XL8'C7100000890ABCDE' ergibt Anzeige 0~Equal
	.	

Die obigen Befehle setzen die Anzeige auf 0~Equal. Der Eingangsoperand hat den Wert -(16^7-1+0,5...), das Ergebnis hat den Wert -16^7. Dieses Ergebnis ist eine *kurze* Gleitpunktzahl. Da der rechte Teil des Ergebnisregisters 0 unverändert bleibt, ist die - hier nur zur Demonstration gezeigte - Interpretation des Ergebnisses als *lange* Gleitpunktzahl arithmetisch nicht korrekt.

Load and Test

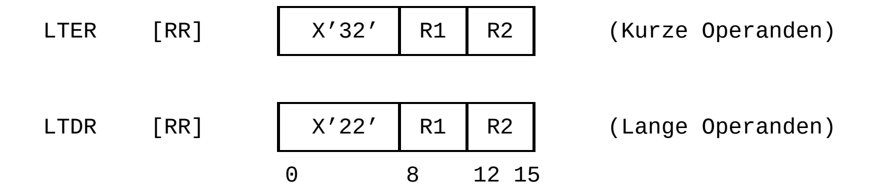
Funktion

Die Befehle LTER und LTDR laden die Gleitpunktzahl im Gleitpunktregister R2 in das Gleitpunktregister R1 und setzen die Anzeige gemäß dem Wert in R1.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Operanden:			
	LTER	R1,R2	R1,R2 =0, 2, 4 oder 6
* lange Operanden:			
	LTDR	R1,R2	R1,R2 =0, 2, 4 oder 6

Maschinenformate



Beschreibung

Die kurze (LTER) bzw. lange (LTDR) Gleitpunktzahl im Gleitpunktregister R2 wird unverändert in das Gleitpunktregister R1 übertragen. Die Anzeige wird gemäß dem Wert der übertragenen Zahl gesetzt. Es findet keine Normalisierung statt.

Anzeige

0~Zero	Mantisse des Ergebnisses ist = 0.
1~Minus	Ergebnis ist < 0.
2~Plus	Ergebnis ist > 0.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßfehler	X’5C’	falsche Gleitpunktregisterangabe

Programmierhinweise

– R1 darf =R2 sein.

– Der Befehl LTER überträgt und testet nur die linken 32 Bit des Gleitpunktregisters R2 und läßt die rechten 32 Bit des Gleitpunkregisters R1 unverändert. Demzufolge kann es geschehen, daß ein LTER Gleichheit anzeigt, wo ein LTDR dies nicht tut.

Multiply

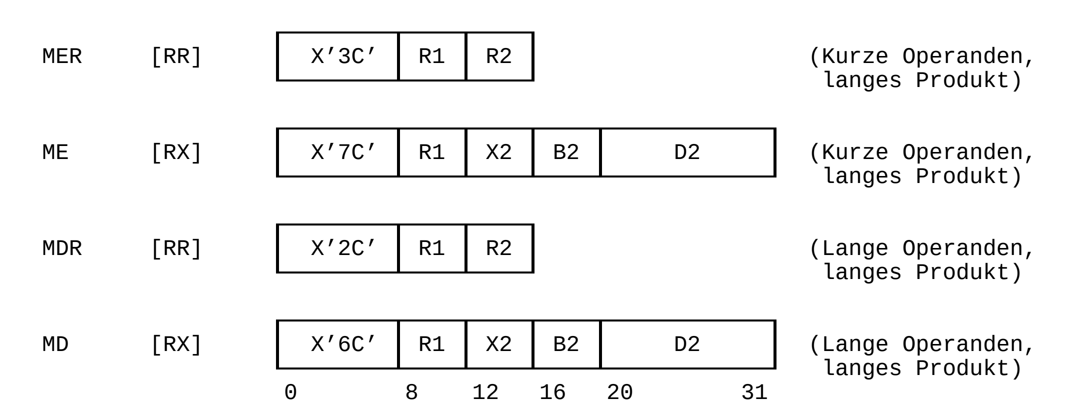
Funktion

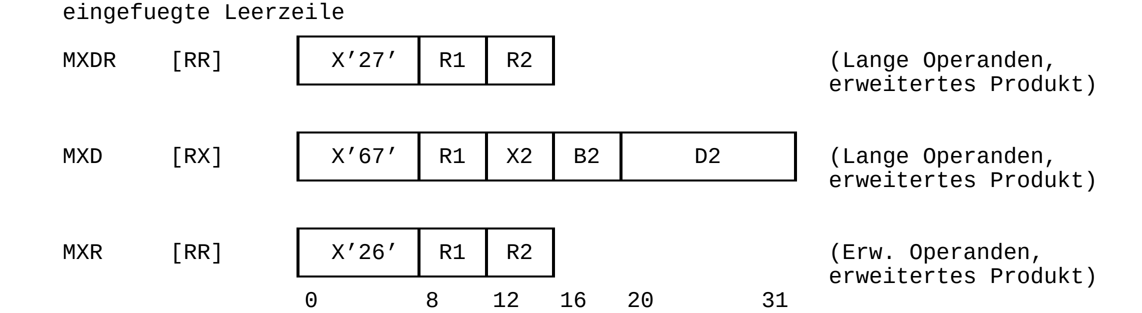
Die Befehle MER, ME, MDR, MD, MXDR, MXD und MXR multiplizieren zwei Gleitpunktzahlen. Das normalisierte Produkt ersetzt den ersten Operanden.
Die Anzeige wird nicht verändert.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurzer Multiplikand und Multiplikator, langes Produkt:			
	MER	R1,R2	R1,R2 =0, 2, 4 oder 6
	ME	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* langer Multiplikand und Multiplikator, langes Produkt:			
	MDR	R1,R2	R1,R2 =0, 2, 4 oder 6
	MD	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* langer Multiplikand und Multiplikator, erweitertes Produkt:			
	MXDR	R1,R2	R1 =0 oder 4 und
*			R2 =0, 2, 4 oder 6
	MXD	R1,D2(X2,B2)	R1 =0 oder 4 und
* erweiterter Multiplikand und Multiplikator, erweitertes Produkt:			
	MXR	R1,R2	R1,R2 =0 oder 4

Maschinenformate





Beschreibung

Der erste Gleitpunktoperand ist der Multiplikand, der zweite Gleitpunktoperand ist der Multiplikator. Das (normalisierte) Produkt ersetzt den ersten Operanden.

Bei den Befehlen MER und ME haben Multiplikand und Multiplikator 6 Sedezimalstellen, bei MDR, MD, MXDR und MXD haben sie 14 und bei MXR haben sie 28 Sedezimalstellen. Das Produkt hat bei MER, ME, MDR und MD 14 Sedezimalstellen, bei MXDR, MXD und MXR hat es 28 Sedezimalstellen.

Zunächst werden Multiplikand und Multiplikator normalisiert. Die Normalisierung geschieht intern, die Eingangs-Operanden werden nicht verändert.

Die Charakteristiken der beiden Operanden werden addiert; die beiden (normaliserten) Mantissen werden multipliziert; das resultierende Produkt bildet mit der Charakteristiksumme minus 64 und dem algebraisch ermittelten Vorzeichen ein Zwischenergebnis. Die Mantisse des Zwischenergebnisses ist exakt. Wenn sie eine führende Sedezimal-Null enthält, wird sie um eine Sedezimalstelle nach links geschoben und die Charakteristik um 1 vermindert. Abschließend wird das Endergebnis erzeugt, indem das Zwischenergebnis bei ME und MER mit 2 sedezimalen Nullen auf 14 Sedezimalstellen verlängert und bei den übrigen Befehlen auf 14 bzw. 28 Sedezimalstellen gekürzt wird.

Exponenten-Überlauf tritt ein, wenn die Charakteristik des Endergebnisses größer als 127 wird und seine Mantisse von Null verschieden ist. Es erfolgt dann eine Programmunterbrechung: Mantisse und Vorzeichen sind korrekt, aber die Charakteristik des Ergebnisses ist um 128 zu klein. Bei MXDR, MXD und MXR ist ggf. auch die Charakteristik des unteren Teils um 128 zu klein.

Exponenten-Überlauf kann erst beim Endergebnis auftreten, nicht schon, wenn beim Zwischenergebnis eine Charakteristik überläuft.

Exponenten-Unterlauf tritt ein, wenn die Charakteristik des Endergebnisses kleiner als 0 wird und seine Mantisse von Null verschieden ist. Wenn in diesem Falle das Bit für Exponenten-Unterlauf in der Programmaske =1 ist (BS2000-Standard), erfolgt eine Programmunterbrechung:

Mantisse und Vorzeichen sind korrekt, aber die Charakteristik des Ergebnisses ist um 128 zu groß; andernfalls erfolgt keine Programmunterbrechung und als Endergebnis ist eine echte Null erzeugt. Bei MDXR, MXD und MXR wird kein Exponenten-Unterlauf erkannt, wenn nur der untere Teil unterläuft.

Das Vorzeichen des Endergebnisses wird nach den üblichen algebraischen Regeln ermittelt; eine echte Null hat jedoch immer ein positives Vorzeichen.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	ME, MD, MXD: Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	falsche Gleitpunktregisterangabe oder D2(X2,B2) keine (Doppel-)Wortgrenze
Exponenten-Überlauf	X’64’	Produkt-Charakteristik > 127
Exponenten-Unterlauf	X’70’	Produkt-Charakteristik < 0

Programmierhinweise

- Die Vertauschung von Multiplikand und Multiplikator verändert in keinem Falle das Ergebnis.
- Bei MER und ME werden für die Mantissenmultiplikation die rechten 32 Bit der verwendeten Gleitpunktregister ignoriert. Allerdings werden die rechten 32 Bit von R1 bei diesen Befehlen durch das Produkt überschrieben.
- Bei MXDR und MXD wird der Inhalt des Gleitpunktregisters R1+2 für die Mantissenmultiplikation ignoriert. Sein Inhalt wird aber durch den unteren Teil des Produkts überschrieben. Bei MXDR wird auch der Inhalt des Gleitpunktregisters R2+2 ignoriert.
- Durch das BS2000 wird das Maskenbit für Exponenten-Unterlauf in der Programmaske mit 1 vorbesetzt, so daß unter den oben beschriebenen Bedingungen eine Programmunterbrechung erfolgt. Mit dem Befehl SPM kann das Anwenderprogramm jedoch die Vorbesetzung ändern.
- R2 darf =R1 sein.
- Bei den Befehlen MER, ME, MXDR und MXD ist das Ergebnis exakt, bei MDR, MD und MXR können rechtsstehende Sedezimalziffern durch Abschneiden verloren gehen.

Beispiel

Name	Operation	Operanden	
	.		
FLNO1	DC	EE2'2.56'	=X'43100000'
FLNO2	DC	ES4'-16'	=X'C6000010'
FLNO3	DC	D'-4096'	=X'C410000000000000'
	.		
	.		
	.		
	LE	6,FLNO1	
	ME	6,FLNO2	
	CD	6,FLNO3	ergibt Anzeige 0~Equal
	.		

Der Befehl ME erzeugt im Gleitpunktregister 6 den Wert
D'4096'= X'C410000000000000'; der Befehl CD setzt die Anzeige auf 0~Equal.
In dem Beispiel ist von den Möglichkeiten des Assembler zur Datenerklärung von Gleitpunktzahlen Gebrauch gemacht. Der Konstantentyp E in den Datenerklärungen von FLNO1 und FLNO2 bewirkt, daß der Assembler kurze Gleitpunktzahlen generiert, der Konstantentyp D bewirkt die Generierung einer langen Gleitpunktzahl. Der Exponentenfaktor "E2" (bei FLNO1) bewirkt die Multiplikation des Arguments 2.56 mit 10^2 und der Skalenfaktor "S4" (bei FLNO2) erzeugt eine um 4 Sedezimalstellen nach rechts versetzte Mantisse. Weitere Möglichkeiten für Gleitpunkt-Datenerklärungen sind im Handbuch ASSEMBH, Beschreibung [1] beschrieben.

Subtract Normalized

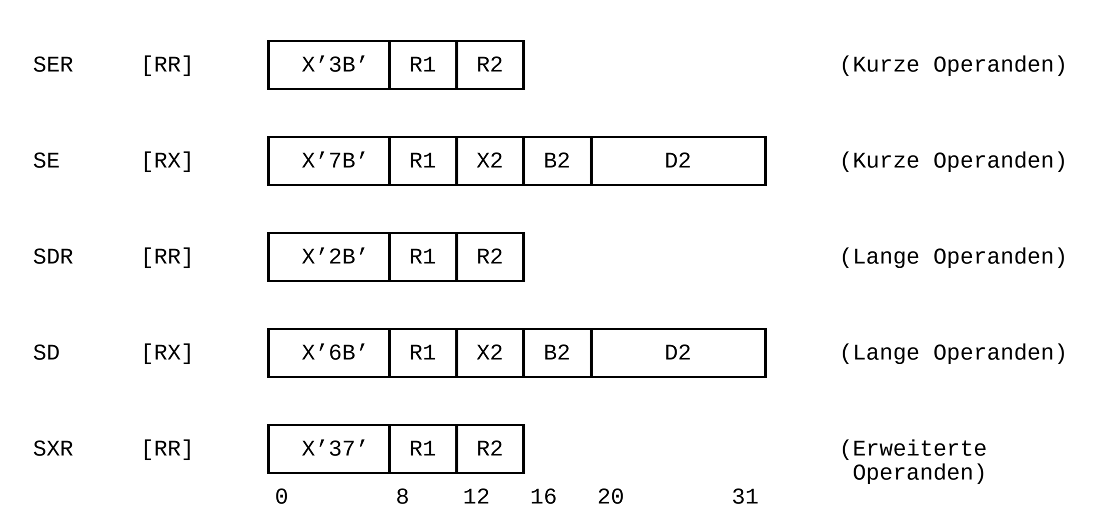
Funktion

Die Befehle SER, SE, SDR, SD und SXR subtrahieren zwei Gleitpunktzahlen. Die normalisierte Differenz ersetzt den ersten Operanden.
Die Anzeige wird gemäß dem Wert der Differenz gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Operanden, kurze Differenz:			
	SER	R1,R2	R1,R2 =0, 2, 4 oder 6
	SE	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* lange Operanden, lange Differenz:			
	SDR	R1,R2	R1,R2 =0, 2, 4 oder 6
	SD	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* erweiterte Operanden, erweiterte Differenz:			
	SXR	R1,R2	R1,R2 =0 oder 4

Maschinenformate



Beschreibung

Zunächst werden die Charakteristiken der beiden Operanden verglichen; die Mantisse des Operanden mit der kleineren Charakteristik wird um die Differenz der Charakteristiken nach rechts geschoben und seine Charakteristik um denselben Betrag erhöht, so daß die Charakteristiken gleich werden. Die zuletzt hinausgeschobene Sedezimalziffer wird als Schutzziffer aufbewahrt. Die Schutzziffer des anderen Operanden - oder beider Operanden, wenn vor der Subtraktion die Charakteristiken gleich waren - wird =0 gesetzt.

Anschließend werden die beiden Mantissen einschließlich der Schutzziffern unter Berücksichtigung der Vorzeichen subtrahiert (Operand1-Mantisse minus Operand2-Mantisse). Die Differenz bildet ein Zwischenergebnis. Das Zwischenergebnis besteht beim kurzen Format aus 7, beim langen Format aus 15 und beim erweiterten Format aus 29 Sedezimalstellen.
Wenn ein Überlauf aufgetreten ist, wird das Zwischenergebnis um eine Sedezimalstelle nach rechts verschoben, dann in die links freigewordene Sedezimalstelle eine 1_{16} eingesetzt und die Charakteristik um 1 erhöht.

Signifikanz tritt ein, wenn das Zwischenergebnis einschließlich Schutzziffer =0 ist. Wenn in diesem Fall das Bit für Signifikanz in der Programmaske =1 ist (BS2000-Standard), erfolgt eine Programmunterbrechung, andernfalls erfolgt keine Programmunterbrechung und als Endergebnis wird eine echte Null erzeugt.

Wenn das Zwischenergebnis einschließlich Schutzziffer $\neq$0 ist, wird es normalisiert, d.h. so lange nach links geschoben, bis die höchstwertige Sedezimalstelle von 0_{16} verschieden ist. Rechts freiwerdende Sedezimalstellen werden mit 0_{16} gefüllt. Die Charakteristik wird um die Anzahl der geschobenen Sedezimalstellen vermindert.
Zuletzt wird das normalisierte Zwischenergebnis auf 6 bzw. 14 bzw. 28 Sedezimalstellen gekürzt und zusammen mit der zuvor ermittelten Charakteristik zum Endergebnis gemacht. Beim erweiterten Format wird im unteren Teil der Gleitpunkt-Differenz eine Charakteristik erzeugt, die um 14 kleiner ist als die des oberen Teils und das Vorzeichen des unteren Teils wird gleich dem des oberen Teils gesetzt.

Exponenten-Überlauf tritt ein, wenn die Charakteristik des Endergebnisses größer als 127 wird. Es erfolgt dann eine Programmunterbrechung: Vorzeichen und Mantisse sind korrekt, aber die Ergebnis-Charakteristik(en) sind um 128 zu klein.

Exponenten-Unterlauf tritt ein, wenn die Charakteristik des Endergebnisses kleiner als 0 wird. Wenn in diesem Falle das Bit für Exponenten-Unterlauf in der Programmaske =1 ist (BS2000-Standard), erfolgt eine Programmunterbrechung: Vorzeichen und Mantisse sind korrekt, aber die Ergebnis-Charakteristik(en) sind um 128 zu groß; andernfalls erfolgt keine Programmunterbrechung und als Endergebnis wird eine echte Null erzeugt.
Beim Befehl SXR tritt Exponenten-Unterlauf nicht ein, wenn nur beim unteren Teil des Ergebnisses die Charakteristik kleiner als 0 wird. In diesem Falle ist dessen Charakteristik um 128 zu groß gesetzt.

Anzeige

0~Zero	Die Mantisse des Ergebnisses ist = 0;
1~Minus	Ergebnis ist < 0.
2~Plus	Ergebnis ist > 0.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	SE, SD: Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	falsche Gleitpunktregisterangabe oder D2(X2,B2) keine (Doppel-)Wortgrenze
Exponenten-Überlauf	X’64’	Charakteristik der Differenz > 127
Signifikanz	X’6C’	Mantisse =0, Charakteristik 0 und Maskenbit für Signifikanz =1
Exponenten-Unterlauf	X’70’	Charakteristik der Differenz < 0

Programmierhinweise

– Die normalisierende Subtraktion normalisiert die Differenz, aber nicht die Eingangsoperanden.

– Durch das BS2000 werden die Bit für Exponenten-Unterlauf und Signifikanz in der Programmaske mit 1 vorbesetzt, so daß in den oben genannten Fällen eine Programmunterbrechung eintritt. Mit dem Befehl SPM (Setzen Programmmaske) kann ein Anwenderprogramm die Vorbesetzung ändern.

– Bei SE und SER werden die rechten 32 Bit der verwendeten Gleitpunktregister ignoriert und bleiben unverändert.

– R2 darf =R1 sein; das Ergebnis ist dann eine echte Null.

Beispiel

Name	Operation	Operanden
	.	
	DS	0F
FLNO1	DC	X’46100000’
FLNO2	DC	X’40200000’
	.	
	.	
	.	
	LE	0,FLNO1
	SE	0,FLNO2
	.	

Das Endergebnis im Gleitpunktregister lautet: X’45FFFFFE’ und die Anzeige ist auf 2~Plus gesetzt.
Nach dem Charakteristik-Abgleich hatte der zweite Operand den Wert X’46000000’ und die Schutzziffer $=2_{16}$. Das Zwischenergebnis betrug X’460FFFFFE’.

Store

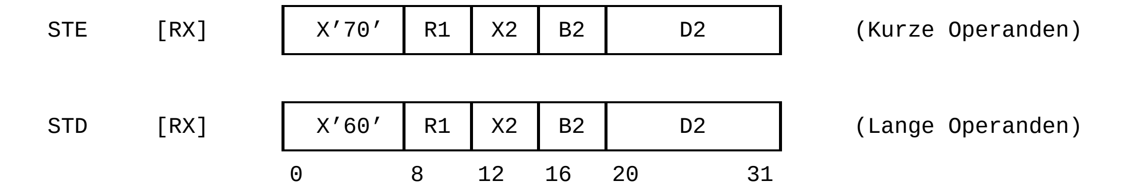
Funktion

Die Befehle STE und STD speichern die Gleitpunktzahl im Gleitpunktregister R1 in ein Hauptspeicherfeld.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Operanden:			
	STE	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* lange Operanden:			
	STD	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und

Maschinenformate



Beschreibung

Die kurze bzw. lange Gleitpunktzahl im Gleitpunktregister R1 wird in das Wort bzw. Doppelwort an der Hauptspeicherstelle D2(X2,B2) gespeichert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X'48'	Schreibzugriff auf Operand2 unmöglich
Adreßfehler	X'5C'	falsche Gleitpunktregisterangabe oder D2(X2,B2) keine (Doppel-)Wortgrenze

Subtract Unnormalized

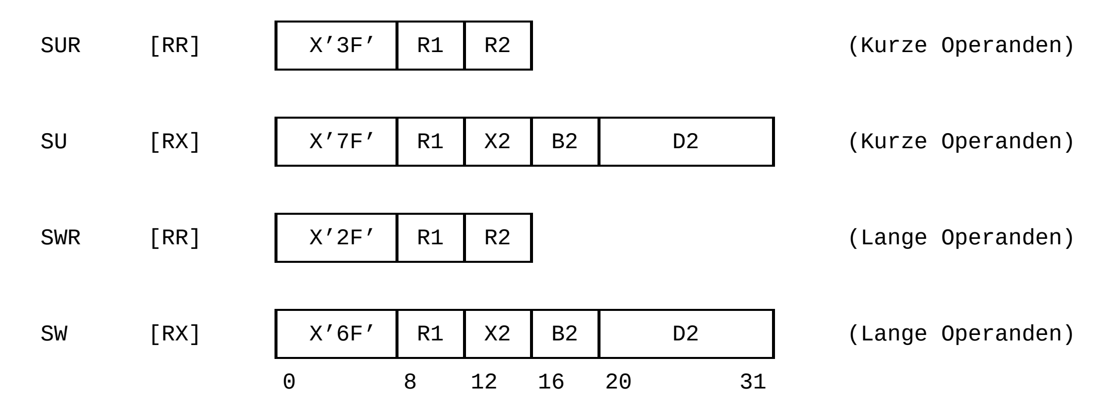
Funktion

Die Befehle SUR, SU, SWR und SW subtrahieren zwei Gleitpunktzahlen. Die Differenz ersetzt den ersten Operanden; sie wird nicht normalisiert. Die Anzeige wird gemäß dem Wert der Differenz gesetzt.

Assemblerformate

Name	Operation	Operanden	Bemerkungen
* kurze Operanden, kurze Differenz:			
	SUR	R1,R2	R1,R2 =0, 2, 4 oder 6
	SU	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und
* lange Operanden, lange Differenz:			
	SWR	R1,R2	R1,R2 =0, 2, 4 oder 6
	SW	R1,D2(X2,B2)	R1 =0, 2, 4 oder 6 und

Maschinenformate



Beschreibung

Zunächst werden die Charakteristiken der beiden Operanden verglichen; die Mantisse des Operanden mit der kleineren Charakteristik wird um die Differenz der Charakteristiken nach rechts geschoben und seine Charakteristik um denselben Betrag erhöht, so daß die Charakteristiken gleich werden. Die zuletzt hinausgeschobene Sedezimalziffer wird als Schutzziffer aufbewahrt. Die Schutzziffer des anderen Operanden - oder beider Operanden, wenn vor der Subtraktion die Charakteristiken gleich waren - wird =0 gesetzt.

Anschließend werden die beiden Mantissen einschließlich der Schutzziffern unter Berücksichtigung der Vorzeichen subtrahiert (Operand1-Mantisse minus Operand2-Mantisse). Die Differenz bildet ein Zwischenergebnis. Das Zwischenergebnis besteht beim kurzen Format aus 7 und beim langen Format aus 15 Sedezimalstellen.

Wenn ein Überlauf aufgetreten ist, wird das Zwischenergebnis um eine Sedezimalstelle nach rechts geschoben, dann in die links freigewordene Sedezimalstelle eine 1_{16} eingesetzt und die Charakteristik um 1 erhöht.

Signifikanz tritt ein, wenn das Zwischenergebnis ausschließlich Schutzziffer =0 ist. Wenn in diesem Fall das Bit für Signifikanz in der Programmaske =1 ist (BS2000-Standard), erfolgt eine Programmunterbrechung, andernfalls erfolgt keine Programmunterbrechung und als Endergebnis wird eine echte Null erzeugt.

Das Zwischenergebnis wird ohne vorherige Normalisierung auf 6 bzw. 14 Sedezimalstellen gekürzt und zusammen mit der zuvor ermittelten Charakteristik zum Endergebnis gemacht.

Exponenten-Überlauf tritt ein, wenn die Charakteristik des Endergebnisses größer als 127 wird. Es erfolgt dann eine Programmunterbrechung: Vorzeichen und Mantisse sind korrekt, aber die Ergebnis-Charakteristik(en) sind um 128 zu klein.

Exponenten-Unterlauf kann nicht eintreten.

Anzeige

0~Zero	Die Mantisse des Ergebnisses ist = 0; das Vorzeichen ist positiv.
1~Minus	Ergebnis ist < 0.
2~Plus	Ergebnis ist > 0.
3	Nicht verwendet.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	SU, SW: Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	falsche Gleitpunktregisterangabe oder D2(X2,B2) keine (Doppel-)Wortgrenze
Exponenten-Überlauf	X’64’	Differenz-Charakteristik > 127
Signifikanz	X’6C’	Mantisse =0, Charakteristik 0 und Maskenbit für Signifikanz =1

Programmierhinweise

- Durch das BS2000 wird das Bit für Signifikanz in der Programmaske mit 1 vorbesetzt, so daß in dem oben genannten Fall eine Programmunterbrechung eintritt. Mit dem Befehl SPM (Setzen Programmaske) kann ein Anwenderprogramm die Vorbesetzung ändern.
- Bei SU und SUR werden die rechten 32 Bit der verwendeten Gleitpunktregister ignoriert und bleiben unverändert.
- Die nicht-normalisierende Subtraktion ist bis auf folgende Unterschiede gleichwertig zur normalisierenden Subtraktion:
  - Das Ergebnis wird nicht normalisiert.
  - Exponenten-Unterlauf kann nicht eintreten.
  - Die Schutzziffer wird nicht zur Ermittlung von Signifikanz herangezogen.
- Für erweiterte Gleitpunktoperanden gibt es zwar einen Befehl zur normalisierenden Subtraktion (SXR), aber keinen Befehl zur nichtnormalisierenden Subtraktion.

Beispiel

Name	Operation	Operanden
	.	
	DS	0D
FLNO1	DC	X’4001111111111111’
FLNO2	DC	X’3F11111111111101’
	.	
	.	
	.	
	LD	2,FLNO1
	SW	2,FLNO2
	.	

Das Ergebnis obiger Befehle hängt vom Wert des Bit für Signifikanz in der Programmmaske ab: wenn dieses Bit =1 ist (BS2000-Standard), lautet das Endergebnis X’4000000000000000’, und es erfolgt eine Programmunterbrechung wegen Signifikanz, andernfalls lautet das Endergebnis X’000000000000000000’ (echte Null) ohne Programmunterbrechung. In beiden Fällen ist die Anzeige auf 0~Zero gesetzt.
Nach dem Charakteristik-Abgleich hatte der zweite Operand den Wert X’4001111111111110’ und die Schutzziffer $=1_{16}$. Das Zwischenergebnis betrug X’400000000000000F’. Man beachte, daß auch bei nicht-normalisierender Subtraktion (und Addition) die Schutzziffern subtrahiert (bzw. addiert) werden.

6 ESA-Befehle

Überblick

Die ESA-Befehle unterstützen die Erweiterung der virtuellen Adreßräume auf den ESA-Anlagen.

a) Zugriffsregister versorgen (CPYA, EAR, SAR, LAM, STAM, LAE).

b) AR-/ASC-Modus abfragen (IAC).
AR-/ASC-Modus setzen oder rücksetzen (SAC).

c) Überprüfen der Zugriffsregister-Adreßumsetzung (TAR).

Auf **ESA-Anlagen** (Enterprise System Architecture) werden neben dem **Programmraum** (program space), der dem bisherigen Adreßraum entspricht, weitere Adreßräume für Daten zur Verfügung gestellt. Diese **Datenräume** (data spaces) haben, wie der Programmraum, virtuelle Adressen (Adresse 0 bis zu 2 Giga-Byte). Sie können nur Daten (oder als Daten abgelegten Programmcode) enthalten, Programmcode kann in einem Datenraum nicht ausgeführt werden. Ihre eindeutige Ansprache ist über die **SPID** (space identification) oder über einen bzw. mehrere **ALET**s (access list entry token) möglich. Die SPID wird beim Anlegen eines Datenraumes vergeben und ist systemweit eindeutig und bekannt. ALETs verweisen nur innerhalb des Programms eindeutig auf einen Datenraum. Für die Adressierung mit ALETs wurden die **Zugriffsregister** (siehe auch 2.2.2) als weiterer Registersatz parallel zu den Mehrzweckregistern eingeführt. In den Zugriffsregistern sind die ALETs enthalten. Wenn der **AR-Modus** (access register mode) eingeschaltet ist, werden bei der Adreßumsetzung in einem Maschinenbefehl die Zugriffsregister mit ausgewertet und so Daten in einem Datenraum adressiert.
Nur Programme, die auf ESA-Anlagen und in einer BS2000-Version ≥ V11 laufen und die ESA-Befehle einsetzen, können Daten in einem solchen Datenraum ablegen. Siehe Handbuch "Makroaufrufe an den Ablaufteil" [3].
Auf ESA-Anlagen können Sie zum einen mit 24-Bit-Adressen oder 31-Bit-Adressen arbeiten und zum anderen mit Datenräumen oder nur im Programmraum. Deshalb gibt es auf ESA-Anlagen einen weiteren Adressierungsmodus, den AR-Modus. Unabhängig vom AR-Modus können Sie die XS-Fähigkeit von ESA-Anlagen nutzen. Der Programmraum und jeder Datenraum, der angelegt wurde, kann entweder nur den unteren Adreßraum, oder den unteren und oberen Adreßraum nutzen. Die Unterstützung von XS-Programmen ist im Handbuch "Einführung in die XS-Programmierung" [2] beschrieben.

AR-Modus

Der AR-Modus (access register mode) ist ein Teil des **ASC-Modus** (address space control mode) und entscheidet über die Auswertung der Zugriffsregister bei der Adreßumsetzung:

- Wenn der AR-Modus eingeschaltet ist, werden die Zugriffsregister bei der Adressierung mit ausgewertet und damit können Adressen in Datenräumen angesprochen werden.
  Der Wert 0 in einem Zugriffsregister hat dabei eine spezielle Bedeutung:
  Mit dem Wert 0 in einem Zugriffsregister kann im AR-Modus der Programmraum adressiert werden. Mit diesem Wert sind die Zugriffsregister beim Start eines Programmes vorbelegt.
- Wenn der AR-Modus ausgeschaltet ist, werden die Zugriffsregister nicht ausgewertet und nur Adressen im Programmraum können angesprochen werden. Das Programm arbeitet wie bisherige Programme auf Nicht-ESA-Anlagen.

Information über den AR-Modus erhalten Sie durch die Abfrage des ASC-Modus mit dem Befehl IAC.
Mit dem Befehl SAC können Sie den AR-Modus ein- und ausschalten.

Hinweise

Die ESA-Programmierung wird durch folgende BS2000-Makros unterstützt:
(siehe Handbuch "Makroaufrufe an den Ablaufteil" [3])

- DSPSRV Datenraum erzeugen und freigeben
- ALESRV Programm mit Datenraum verbinden oder lösen
- ALINF Informationen über Zugriffslisten anfordern

Bei der Adressierung über Zugriffsregister ist zu beachten:
Das zum Zugriffsregister gehörende Mehrzweckregister ist als Basisregister zu verwenden. Wird ein Mehrzweckregister z.B. als Indexregister verwendet, so wird das entsprechende Zugriffsregister ignoriert. Wird in einem Befehl ein Mehrzweckregister als Basisregister spezifiziert, dann muß im AR-Modus das entsprechende Zugriffsregister richtig versorgt sein.
Wegen der strengen Zuordnung von Zugriffsregistern zu Basisregistern dürfen Index- und Basisregister im AR-Modus nicht vertauscht werden.

Copy Access Register

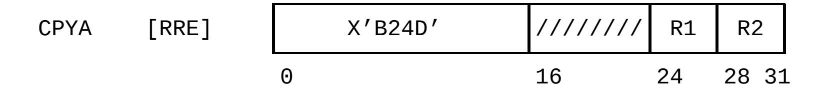
Funktion

Der Befehl CPYA überträgt den Inhalt aus einem Zugriffsregister in ein Zugriffsregister. Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	CPYA	R1,R2	

Maschinenformat



Beschreibung

Der Inhalt des Zugriffsregisters R2 wird in das Zugriffsregister R1 übertragen.

Die Bitstellen 16 bis 23 des Befehls werden ignoriert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine.

Extract Access Register

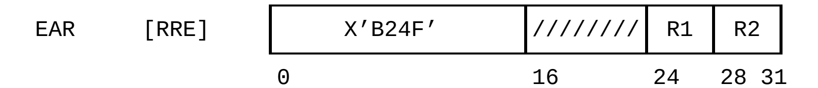
Funktion

Der Befehl EAR überträgt den Inhalt aus einem Zugriffsregister in ein Mehrzweckregister.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	EAR	R1,R2	

Maschinenformat



Beschreibung

Der Inhalt des Zugriffsregisters R2 wird in das Mehrzweckregister R1 übertragen.

Die Bitstellen 16 bis 23 des Befehls werden ignoriert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine.

Insert Address Space Control

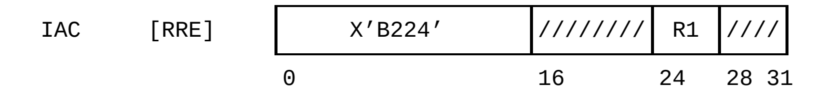
Funktion

Der Befehl IAC überträgt den momentanen Wert des ASC-Modus in ein Mehrzweckregister.
Die Anzeige wird gemäß dem Wert des ASC-Modus gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	IAC	R1	

Maschinenformat



Beschreibung

Den ASC-Modus (primary space mode oder access register mode) können Sie entweder im Register R1 oder über die Anzeige abfragen.

Bit 16 und Bit 17 (address space control bits ≙ ASC-Modus) des aktuellen PSW werden in umgekehrter Reihenfolge in die Bitstellen 22 und 23 des Mehrzweckregisters R1 übertragen, d.h. Bit 16 wird in die Bitstelle 23 und Bit 17 in die Bitstelle 22 des Registers R1 übertragen. Die Bitstellen 16 bis 21 des Registers R1 werden auf 0 gesetzt, die Bitstellen 0 bis 15 und 24 bis 31 des Registers bleiben unverändert.

Die Bitstellen 16 bis 23 und 28 bis 31 des Befehls werden ignoriert.

Anzeige

0 primary space mode (PSW-Bit 16 und PSW-Bit 17 =0)
2 access register mode (PSW-Bit 16 =0 und PSW-Bit 17 =1)

Programmunterbrechungen

Keine.

Load Address Extended

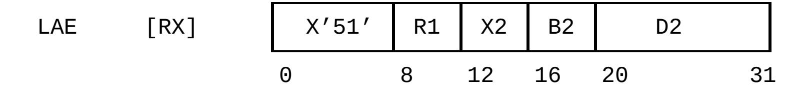
Funktion

Der Befehl LAE lädt ein Mehrzweckregister mit einer Adresse und das korrespondierende Zugriffsregister mit einem Wert.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	LAE	R1,D2(X2,B2)	

Maschinenformat



Beschreibung

Die Adresse D2(X2,B2) wird in das Mehrzweckregister R1 geladen. Die Adresse wird logisch als Summe aus den Adressen in den Mehrzweckregistern X2 und B2 und dem Binärwert des 12 Bit langen D2-Feldes berechnet, wobei keine Vorzeichen berücksichtigt werden und etwaiger Übertrag über die höchstwertige Binärstelle ignoriert wird.
Wenn X2=0 ist, wird der Inhalt des Registers X2, wenn B2=0 ist, wird der Inhalt des Registers B2 *nicht* mitsummiert.

Im 24-Bit-Adressierungsmodus werden vom Inhalt der Mehrzweckregister B2 und X2 nur die niedrigstwertigen 24 Bit zur Summenbildung verwendet; die Summe wird in die Bitstellen 8 bis 31 des Mehrzweckregisters R1 eingetragen und die Bitstellen 0 bis 7 von R1 werden auf 0 gesetzt.
Im 31-Bit-Adressierungsmodus werden von B2 und X2 nur die niedrigstwertigen 31 Bit zur Summenbildung verwendet; die Summe wird in die Bitstellen 1 bis 31 des Mehrzweckregisters R1 eingetragen und das Bit 0 auf 0 gesetzt.

Das korrespondierende Zugriffsregister R1 wird mit einem Wert geladen, der vom AR-Modus, dem momentanen Wert der Bitstellen 16 und 17 (address space control bits) des PSW abhängt. Wenn die Bitstellen 16 und 17 den binären Wert 01 haben, d.h. der AR-Modus (access register mode) ist eingeschaltet, hängt der Wert im Zugriffsregister auch davon ab, ob das B2-Feld =0 oder ≠0 ist (siehe folgende Tabelle).

PSW-Bit 16 und 17	Modus	Wert im Zugriffsregister R1
00	primary space mode	X'00000000' d.h. die Bitstellen 0 bis 31 sind =0
01	access register mode	X'00000000', wenn das B2-Feld =0 ist. Ist das B2-Feld 0, so wird der Inhalt des Zugriffsregisters B2 in das Zugriffsregister R1 übertragen. Die Bitstellen 0 bis 6 des Zugriffsregisters B2 müssen =0 sein, sonst sind die Ergebnisse im Mehrzweckregister R1 und im Zugriffsregister R1 nicht vorhersagbar.

Es erfolgt kein Speicherzugriff auf die resultierende Adresse.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine.

Load Access Multiple

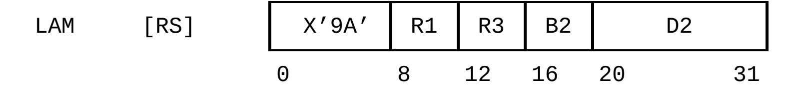
Funktion

Der Befehl LAM lädt aus dem Hauptspeicher bis zu 16 aufeinanderfolgende Worte in aufeinanderfolgende Zugriffsregister.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	LAM	R1,R3,D2(B2)	D2(B2): Wortgrenze

Maschinenformat



Beschreibung

Die aufeinanderfolgenden Zugriffsregister, beginnend mit R1 und endend mit R3, werden mit aufeinanderfolgenden Worten geladen, deren erstes mit D2(B2) adressiert ist.

Ist R1=R3, so wird nur ein Register (R1) geladen. Ist R3 kleiner als R1, so wird von R1 aufwärts bis zum Zugriffsregister 15 und vom Zugriffsregister 0 bis zum und einschließlich R3 geladen.

Befehl	Operand1	Operand2
LAM	Inhalt von Zugriffsregister R1 bis R3	Mit D2(B2) adressierte Wortfolge; Wortanzahl =R3-R1+1, wenn R3≥R1 =R3-R1+17, wenn R3<R1

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler	X’48’	Lesezugriff auf Operand2 unmöglich
Adreßfehler	X’5C’	D2(B2) keine Wortgrenze

Set Address Space Control

Funktion

Der Befehl SAC schaltet den AR-Modus (access register mode) ein oder aus.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	SAC	D2(B2)	

Maschinenformat

SAC [S]

X'B219'	B2	D2
0	16	20 31

Beschreibung

Mit den Bitstellen 20 bis 23 des D2-Feldes oder des Registers B2 können Sie die address space control bit im PSW (Bit 16 und Bit 17) setzen und damit den AR-Modus (access register mode) ein- oder ausschalten:

- Direkt im D2-Feld
  Sie setzen die Bitstellen 20 bis 23, wie in folgender Tabelle dargestellt. Die Bitstellen 20 und 21 müssen =0 sein. Die Bitstellen 24 bis 31 werden ignoriert.
- Über das B2-Feld bestimmen Sie ein Mehrzweckregister (MZR)
  Sie laden das Register mit den Werten (Bit 20 bis 23), wie in folgender Tabelle dargestellt. Die Bitstellen 20 und 21 müssen =0 sein. Die Bitstellen 0 bis 19 und 24 bis 31 des Registers werden ignoriert.

D2-Feld/MZR Bitstellen 20,21,22,23	Modus	PSW-Bit 16 und 17
0000	primary space mode	00
0010	access register mode	01

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine.

Programmierhinweis

– Die Werte der Bitstellen 20 bis 23 des D2-Feldes oder des MZR (B2) entsprechen den Werten, die der Befehl IAC in einem MZR abspeichert.

Beispiel

Name	Operation	Operanden	
	.		
	SAC	512	AR-Modus einschalten
	SAC	0	AR-Modus ausschalten
	.		
	.		

Set Access Register

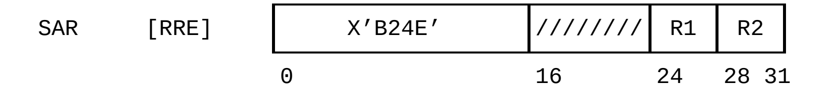
Funktion

Der Befehl SAR überträgt den Inhalt aus einem Mehrzweckregister in ein Zugriffsregister.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	SAR	R1,R2	

Maschinenformat



Beschreibung

Der Inhalt des Mehrzweckregisters R2 wird in das Zugriffsregister R1 übertragen.

Die Bitstellen 16 bis 23 des Befehls werden ignoriert.

Anzeige

Nicht verändert.

Programmunterbrechungen

Keine.

Store Access Multiple

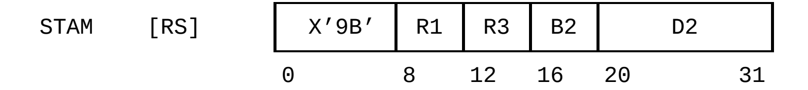
Funktion

Der Befehl STAM speichert die Inhalte von (bis zu 16) aufeinanderfolgenden Zugriffsregistern in aufeinanderfolgende Worte des Hauptspeichers.
Die Anzeige wird nicht verändert.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	STAM	R1,R3,D2(B2)	D2(B2): Wortgrenze

Maschinenformat



Beschreibung

Der Inhalt aufeinanderfolgender Zugriffsregister, beginnend mit R1 und endend mit R3, wird in aufeinanderfolgende Worte des Hauptspeichers übertragen. Das erste Wort ist mit D2(B2) adressiert.
Wenn R1 > R3 ist, wird ab dem Zugriffsregister R1 bis zum Zugriffsregister 15 und dann ab dem Zugriffsregister 0 bis zum Register R3 gespeichert. Wenn R1=R3 ist, wird nur ein Zugriffsregister (R1) gespeichert.

Befehl	Operand1	Operand2
STAM	Inhalt von Zugriffsregister R1 bis R3	mit D2(B2) adressierte Wortfolge Wortanzahl =R3-R1+1, wenn R3≥R1 =R3-R1+17, wenn R3<R1

Anzeige

Nicht verändert.

Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Adreßfehler	X’48’ X’5C’	Schreibzugriff auf Operand2 unmöglich D2(B2) keine Wortgrenze

Test Access Register

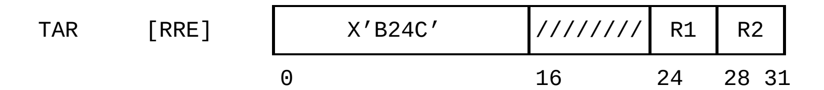
Funktion

Der Befehl TAR überprüft, ob während einer Adreßumsetzung mittels eines Zugriffsregisters (ART, access register translation) eine Ausnahmebedingung aufgetreten ist.
Die Anzeige wird gemäß dem ALET-Wert gesetzt.

Assemblerformat

Name	Operation	Operanden	Bemerkungen
	TAR	R1,R2	

Maschinenformat



Beschreibung

Der Inhalt des Zugriffsregisters R1 (ALET, access list entry token) wird auf Ausnahmebedingungen geprüft, die während der ART (access register translation) erkannt wurden. Der ALET wird geprüft, ob er einen gültigen Eintrag in der Zugriffsliste referenziert oder ob der Inhalt X’00000000’ beträgt.

Wenn R1 =0 ist, wird der Inhalt des Zugriffsregisters 0 in der ART verwendet, anstatt wie üblich X’00000000’.

Die Bitstellen 0 bis 15 des Mehrzweckregisters R2 werden zur Zeit (BS2000 ≥ V11) ignoriert. Die Bitstellen 16 bis 31 des Registers werden ignoriert.

Die Bitstellen 16 bis 23 des Befehls werden ignoriert.

Anzeige

0 ALET (access list entry token) ist X'00000000'.
1 ALET verursacht keine Ausnahmebedingungen in der ART (access register translation).
2 ALET verursacht keine Ausnahmebedingungen in der ART.
3 ALET verursacht Ausnahmebedingungen in der ART.

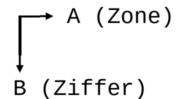
Programmunterbrechungen

Art	Gewicht	Ursachen
Adreßumsetzungsfehler Special Operation Exception	X'48' X'54'	ESA-Funktionen nicht verfügbar

7 Anhang

7.1 EBCDIC-Tabelle (SRV.10)

EBCDIC.SRV.10 (Siemens-Referenz-Version des 8-Bit-Codes)



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	TC7			SP	&	-						{	}	\	0
1	TC1	DC1					/		a	j	˜		A	J		1
2	TC2	DC2		TC9			˜		b	k	s		B	K	S	2
3	TC3	DC3					[		c	l	t		C	L	T	3
4							]		d	m	u		D	M	U	4
5	FE1	NL	FE2						e	n	v		E	N	V	5
6		FE0	TCA						f	o	w		F	O	W	6
7	DEL		ESC	TC4			ß		g	p	x		G	P	X	7
8		CAN							h	q	y		H	Q	Y	8
9		EM						‘	i	r	z		I	R	Z	9
A						!	^	:								
B	FE3				.	$	,	#	Ä		ä					
C	FE4	IS4		DC4	<	*	%	@	Ö		ö					
D	FE5	IS3	TC5	TC8	(	)	_	’	Ü		ü					
E	SO[1]	IS2	TC6		+	;	>	=								
F	SI[1]	IS1	BEL	SUB	|		?	"								

1) Die Steuerzeichen SI und SO entfallen voraussichtlich im 8-Bit-Code.

7.2 Befehle nach mnemotechnischem Code

Mnemo. Code	Op. Code	Bef. Typ	Länge in Byte	An-zeige	Bef. art	Assembler-format
A	5A	RX	4	ja	Allg	R1,D2(X2,B2)
AD	6A	RX	4	ja	Glp	R1,D2(X2,B2)
ADR	2A	RR	2	ja	Glp	R1,R2
AE	7A	RX	4	ja	Glp	R1,D2(X2,B2)
AER	3A	RR	2	ja	Glp	R1,R2
AH	4A	RX	4	ja	Allg	R1,D2(X2,B2)
AL	5E	RX	4	ja	Allg	R1,D2(X2,B2)
ALR	1E	RR	2	ja	Allg	R1,R2
AP	FA	SS	6	ja	Dez	D1(L1,B1),D2(L2,B2)
AR	1A	RR	2	ja	Allg	R1,R2
AU	7E	RX	4	ja	Glp	R1,D2(X2,B2)
AUR	3E	RR	2	ja	Glp	R1,R2
AW	6E	RX	4	ja	Glp	R1,D2(X2,B2)
AWR	2E	RR	2	ja	Glp	R1,R2
AXR	36	RR	2	ja	Glp	R1,R2
BAL	45	RX	4	nein	Allg	R1,D2(X2,B2)
BALR	05	RR	2	nein	Allg	R1,R2
BAS	4D	RX	4	nein	Allg	R1,D2(X2,B2)
BASR	0D	RR	2	nein	Allg	R1,R2
BASSM	0C	RR	2	nein	Allg	R1,R2
BC	47	RX	4	nein	Allg	M1,D2(X2,B2)
BCR	07	RR	2	nein	Allg	M1,R2
BCT	46	RX	4	nein	Allg	R1,D2(X2,B2)
BCTR	06	RR	2	nein	Allg	R1,R2
BSM	0B	RR	2	nein	Allg	R1,R2
BXH	86	RS	4	nein	Allg	R1,R3,D2(B2)
BXLE	87	RS	4	nein	Allg	R1,R3,D2(B2)
C	59	RX	4	ja	Allg	R1,D2(X2,B2)
CD	69	RX	4	ja	Glp	R1,D2(X2,B2)
CDR	29	RR	2	ja	Glp	R1,R2
CDS	BB	RS	4	ja	Allg	R1,R3,D2(B2)
CE	79	RX	4	ja	Glp	R1,D2(X2,B2)
CER	39	RR	2	ja	Glp	R1,R2
CH	49	RX	4	ja	Allg	R1,D2(X2,B2)
CL	55	RX	4	ja	Allg	R1,D2(X2,B2)
CLC	D5	SS	6	ja	Allg	D1(L,B1),D2(B2)
CLCL	0F	RR	2	ja	Allg	R1,R2
CLI	95	SI	4	ja	Allg	D1(B1),I2
CLM	BD	RS	4	ja	Allg	R1,M3,D2(B2)
CLR	15	RR	2	ja	Allg	R1,R2
CP	F9	SS	6	ja	Dez	D1(L1,B1),D2(L2,B2)
CPYA	B24D	RRE	4	nein	ESA	R1,R2
CR	19	RR	2	ja	Allg	R1,R2
CS	BA	RS	4	ja	Allg	R1,R3,D2(B2)
CVB	4F	RX	4	nein	Allg	R1,D2(X2,B2)
CVD	4E	RX	4	nein	Allg	R1,D2(X2,B2)
D	5D	RX	4	nein	Allg	R1,D2(X2,B2)
DD	6D	RX	4	nein	Glp	R1,D2(X2,B2)
DDR	2D	RR	2	nein	Glp	R1,R2
DE	7D	RX	4	nein	Glp	R1,D2(X2,B2)
DER	3D	RR	2	nein	Glp	R1,R2

Mnemo. Code	Op. Code	Bef. Typ	Länge in Byte	An- zeige	Bef. art	Assembler- format
DP	FD	SS	6	nein	Dez	D1(L1,B1),D2(L2,B2)
DR	1D	RR	2	nein	Allg	R1,R2
DXR	B22D	RRE	4	nein	Glp	R1,R2
EAR	B24F	RRE	4	nein	ESA	R1,R2
ED	DE	SS	6	ja	Dez	D1(L,B1),D2(B2)
EDMK	DF	SS	6	ja	Dez	D1(L,B1),D2(B2)
EX	44	RX	4	ja*	Allg	R1,D2(X2,B2) *befehlsabhängig
HDR	24	RR	2	nein	Glp	R1,R2
HER	34	RR	2	nein	Glp	R1,R2
IAC	B224	RRE	4	ja	ESA	R1
IC	43	RX	4	nein	Allg	R1,D2(X2,B2)
ICM	BF	RS	4	ja	Allg	R1,M3,D2(B2)
IPM	B222	RRE	4	nein	Allg	R1
L	58	RX	4	nein	Allg	R1,D2(X2,B2)
LA	41	RX	4	nein	Allg	R1,D2(X2,B2)
LAE	51	RX	4	nein	ESA	R1,D2(X2,B2)
LAM	9A	RS	4	nein	ESA	R1,R3,D2(B2)
LCDR	23	RR	2	ja	Glp	R1,R2
LCER	33	RR	2	ja	Glp	R1,R2
LCR	13	RR	2	ja	Allg	R1,R2
LD	68	RX	4	nein	Glp	R1,D2(X2,B2)
LDR	28	RR	2	nein	Glp	R1,R2
LE	78	RX	4	nein	Glp	R1,D2(X2,B2)
LER	38	RR	2	nein	Glp	R1,R2
LH	48	RX	4	nein	Allg	R1,D2(X2,B2)
LM	98	RS	4	nein	Allg	R1,R3,D2(B2)
LNDR	21	RR	2	ja	Glp	R1,R2
LNER	31	RR	2	ja	Glp	R1,R2
LNR	11	RR	2	ja	Allg	R1,R2
LPDR	20	RR	2	ja	Glp	R1,R2
LPER	30	RR	2	ja	Glp	R1,R2
LPR	10	RR	2	ja	Allg	R1,R2
LR	18	RR	2	nein	Allg	R1,R2
LRDR	25	RR	2	nein	Glp	R1,R2
LRER	35	RR	2	nein	Glp	R1,R2
LTDR	22	RR	2	ja	Glp	R1,R2
LTER	32	RR	2	ja	Glp	R1,R2
LTR	12	RR	2	ja	Allg	R1,R2
M	5C	RX	4	nein	Allg	R1,D2(X2,B2)
MC	AF	SI	4	ja	Allg	D1(B1),I2
MD	6C	RX	4	nein	Glp	R1,D2(X2,B2)
MDR	2C	RR	2	nein	Glp	R1,R2
ME	7C	RX	4	nein	Glp	R1,D2(X2,B2)
MER	3C	RR	2	nein	Glp	R1,R2
MH	4C	RX	4	nein	Allg	R1,D2(X2,B2)
MP	FC	SS	6	nein	Dez	D1(L1,B1),D2(L2,B2)
MR	1C	RR	2	nein	Allg	R1,R2
MVC	D2	SS	6	nein	Allg	D1(L,B1),D2(B2)
MVCL	0E	RR	2	ja	Allg	R1,R2
MVI	92	SI	4	nein	Allg	D1(B1),I2
MVN	D1	SS	6	nein	Allg	D1(L,B1),D2(B2)
MVO	F1	SS	6	nein	Allg	D1(L1,B1),D2(L2,B2)
MVZ	D3	SS	6	nein	Allg	D1(L,B1),D2(B2)
MXD	67	RX	4	nein	Glp	R1,D2(X2,B2)

Mnemo. Code	Op. Code	Bef. Typ	Länge in Byte	An- zeige	Bef. art	Assembler- format
MXDR	27	RR	2	nein	Glp	R1,R2
MXR	26	RR	2	nein	Glp	R1,R2
N	54	RX	4	ja	Allg	R1,D2(X2,B2)
NC	D4	SS	6	ja	Allg	D1(L,B1),D2(B2)
NI	94	SI	4	ja	Allg	D1(B1),I2
NR	14	RR	2	ja	Allg	R1,R2
O	56	RX	4	ja	Allg	R1,D2(X2,B2)
OC	D6	SS	6	ja	Allg	D1(L,B1),D2(B2)
OI	96	SI	4	ja	Allg	D1(B1),I2
OR	16	RR	2	ja	Allg	R1,R2
PACK	F2	SS	6	nein	Allg	D1(L1,B1),D2(L2,B2)
S	5B	RX	4	ja	Allg	R1,D2(X2,B2)
SAC	B219	S	4	nein	ESA	D2(B2)
SAR	B24E	RRE	4	nein	ESA	R1,R2
SD	6B	RX	4	ja	Glp	R1,D2(X2,B2)
SDR	2B	RR	2	ja	Glp	R1,R2
SE	7B	RX	4	ja	Glp	R1,D2(X2,B2)
SER	3B	RR	2	ja	Glp	R1,R2
SH	4B	RX	4	ja	Allg	R1,D2(X2,B2)
SL	5F	RX	4	ja	Allg	R1,D2(X2,B2)
SLA	8B	RS	4	ja	Allg	R1,D2(B2)
SLDA	8F	RS	4	ja	Allg	R1,D2(B2)
SLDL	8D	RS	4	nein	Allg	R1,D2(B2)
SLL	89	RS	4	nein	Allg	R1,D2(B2)
SLR	1F	RR	2	ja	Allg	R1,R2
SP	FB	SS	6	ja	Dez	D1(L1,B1),D2(L2,B2)
SPM	04	RR	2	ja	Allg	R1
SR	1B	RR	2	ja	Allg	R1,R2
SRA	8A	RS	4	ja	Allg	R1,D2(B2)
SRDA	8E	RS	4	ja	Allg	R1,D2(B2)
SRDL	8C	RS	4	nein	Allg	R1,D2(B2)
SRL	88	RS	4	nein	Allg	R1,D2(B2)
SRP	F0	SS	6	ja	Dez	D1(L1,B1),D2(B2),I3
ST	50	RX	4	nein	Allg	R1,D2(X2,B2)
STAM	9B	RS	4	nein	ESA	R1,R3,D2(B2)
STC	42	RX	4	nein	Allg	R1,D2(X2,B2)
STCK	B205	S	4	ja	Allg	D2(B2)
STCM	BE	RS	4	nein	Allg	R1,M3,D2(B2)
STD	60	RX	4	nein	Glp	R1,D2(X2,B2)
STE	70	RX	4	nein	Glp	R1,D2(X2,B2)
STH	40	RX	4	nein	Allg	R1,D2(X2,B2)
STM	90	RS	4	nein	Allg	R1,R3,D2(B2)
SU	7F	RX	4	ja	Glp	R1,D2(X2,B2)
SUR	3F	RR	2	ja	Glp	R1,R2
SVC	0A	RR	2	nein	Allg	I
SW	6F	RX	4	ja	Glp	R1,D2(X2,B2)
SWR	2F	RR	2	ja	Glp	R1,R2
SXR	37	RR	2	ja	Glp	R1,R2
TAR	B24C	RRE	4	ja	ESA	R1,R2
TM	91	SI	4	ja	Allg	D1(B1),I2
TR	DC	SS	6	nein	Allg	D1(L,B1),D2(B2)
TRT	DD	SS	6	ja	Allg	D1(L,B1),D2(B2)
TS	93	S	4	ja	Allg	D2(B2)
UNPK	F3	SS	6	nein	Allg	D1(L1,B1),D2(L2,B2)

Mnemo. Code	Op. Code	Bef. Typ	Länge in Byte	An- zeige	Bef. art	Assembler- format
X	57	RX	4	ja	Allg	R1,D2(X2,B2)
XC	D7	SS	6	ja	Allg	D1(L,B1),D2(B2)
XI	97	SI	4	ja	Allg	D1(B1),I2
XR	17	RR	2	ja	Allg	R1,R2
ZAP	F8	SS	6	ja	Dez	D1(L1,B1),D2(L2,B2)

7.3 Befehle nach Operationscode

Op. Code	Mnemo. Code	Bef. typ	Länge in Byte	An- zeige	Bef. art	Assembler- format
04	SPM	RR	2	ja	Allg	R1
05	BALR	RR	2	nein	Allg	R1,R2
06	BCTR	RR	2	nein	Allg	R1,R2
07	BCR	RR	2	nein	Allg	M1,R2
0A	SVC	RR	2	nein	Allg	I
0B	BSM	RR	2	nein	Allg	R1,R2
0C	BASSM	RR	2	nein	Allg	R1,R2
0D	BASR	RR	2	nein	Allg	R1,R2
0E	MVCL	RR	2	ja	Allg	R1,R2
0F	CLCL	RR	2	ja	Allg	R1,R2
10	LPR	RR	2	ja	Allg	R1,R2
11	LNR	RR	2	ja	Allg	R1,R2
12	LTR	RR	2	ja	Allg	R1,R2
13	LCR	RR	2	ja	Allg	R1,R2
14	NR	RR	2	ja	Allg	R1,R2
15	CLR	RR	2	ja	Allg	R1,R2
16	OR	RR	2	ja	Allg	R1,R2
17	XR	RR	2	ja	Allg	R1,R2
18	LR	RR	2	nein	Allg	R1,R2
19	CR	RR	2	ja	Allg	R1,R2
1A	AR	RR	2	ja	Allg	R1,R2
1B	SR	RR	2	ja	Allg	R1,R2
1C	MR	RR	2	nein	Allg	R1,R2
1D	DR	RR	2	nein	Allg	R1,R2
1E	ALR	RR	2	ja	Allg	R1,R2
1F	SLR	RR	2	ja	Allg	R1,R2
20	LPDR	RR	2	ja	Glp	R1,R2
21	LNDR	RR	2	ja	Glp	R1,R2
22	LTDR	RR	2	ja	Glp	R1,R2
23	LCDR	RR	2	ja	Glp	R1,R2
24	HDR	RR	2	nein	Glp	R1,R2
25	LRDR	RR	2	nein	Glp	R1,R2
26	MXR	RR	2	nein	Glp	R1,R2
27	MXDR	RR	2	nein	Glp	R1,R2
28	LDR	RR	2	nein	Glp	R1,R2
29	CDR	RR	2	ja	Glp	R1,R2
2A	ADR	RR	2	ja	Glp	R1,R2
2B	SDR	RR	2	ja	Glp	R1,R2
2C	MDR	RR	2	nein	Glp	R1,R2
2D	DDR	RR	2	nein	Glp	R1,R2
2E	AWR	RR	2	ja	Glp	R1,R2
2F	SWR	RR	2	ja	Glp	R1,R2
30	LPER	RR	2	ja	Glp	R1,R2
31	LNER	RR	2	ja	Glp	R1,R2
32	LTER	RR	2	ja	Glp	R1,R2
33	LCER	RR	2	ja	Glp	R1,R2
34	HER	RR	2	nein	Glp	R1,R2
35	LRER	RR	2	nein	Glp	R1,R2
36	AXR	RR	2	ja	Glp	R1,R2
37	SXR	RR	2	ja	Glp	R1,R2
38	LER	RR	2	nein	Glp	R1,R2

Op. Code	Mnemo. Code	Bef. Typ	Länge in Byte	An- zeige	Bef. art	Assembler- format
39	CER	RR	2	ja	Glp	R1,R2
3A	AER	RR	2	ja	Glp	R1,R2
3B	SER	RR	2	ja	Glp	R1,R2
3C	MER	RR	2	nein	Glp	R1,R2
3D	DER	RR	2	nein	Glp	R1,R2
3E	AUR	RR	2	ja	Glp	R1,R2
3F	SUR	RR	2	ja	Glp	R1,R2
40	STH	RX	4	nein	Allg	R1,D2(X2,B2)
41	LA	RX	4	nein	Allg	R1,D2(X2,B2)
42	STC	RX	4	nein	Allg	R1,D2(X2,B2)
43	IC	RX	4	nein	Allg	R1,D2(X2,B2)
44	EX	RX	4	ja*	Allg	R1,D2(X2,B2) *befehlsabhängig
45	BAL	RX	4	nein	Allg	R1,D2(X2,B2)
46	BCT	RX	4	nein	Allg	R1,D2(X2,B2)
47	BC	RX	4	nein	Allg	M1,D2(X2,B2)
48	LH	RX	4	nein	Allg	R1,D2(X2,B2)
49	CH	RX	4	ja	Allg	R1,D2(X2,B2)
4A	AH	RX	4	ja	Allg	R1,D2(X2,B2)
4B	SH	RX	4	ja	Allg	R1,D2(X2,B2)
4C	MH	RX	4	nein	Allg	R1,D2(X2,B2)
4D	BAS	RX	4	nein	Allg	R1,D2(X2,B2)
4E	CVD	RX	4	nein	Allg	R1,D2(X2,B2)
4F	CVB	RX	4	nein	Allg	R1,D2(X2,B2)
50	ST	RX	4	nein	Allg	R1,D2(X2,B2)
51	LAE	RX	4	nein	ESA	R1,D2(X2,B2)
54	N	RX	4	ja	Allg	R1,D2(X2,B2)
55	CL	RX	4	ja	Allg	R1,D2(X2,B2)
56	O	RX	4	ja	Allg	R1,D2(X2,B2)
57	X	RX	4	ja	Allg	R1,D2(X2,B2)
58	L	RX	4	nein	Allg	R1,D2(X2,B2)
59	C	RX	4	ja	Allg	R1,D2(X2,B2)
5A	A	RX	4	ja	Allg	R1,D2(X2,B2)
5B	S	RX	4	ja	Allg	R1,D2(X2,B2)
5C	M	RX	4	nein	Allg	R1,D2(X2,B2)
5D	D	RX	4	nein	Allg	R1,D2(X2,B2)
5E	AL	RX	4	ja	Allg	R1,D2(X2,B2)
5F	SL	RX	4	ja	Allg	R1,D2(X2,B2)
60	STD	RX	4	nein	Glp	R1,D2(X2,B2)
67	MXD	RX	4	nein	Glp	R1,D2(X2,B2)
68	LD	RX	4	nein	Glp	R1,D2(X2,B2)
69	CD	RX	4	ja	Glp	R1,D2(X2,B2)
6A	AD	RX	4	ja	Glp	R1,D2(X2,B2)
6B	SD	RX	4	ja	Glp	R1,D2(X2,B2)
6C	MD	RX	4	nein	Glp	R1,D2(X2,B2)
6D	DD	RX	4	nein	Glp	R1,D2(X2,B2)
6E	AW	RX	4	ja	Glp	R1,D2(X2,B2)
6F	SW	RX	4	ja	Glp	R1,D2(X2,B2)
70	STE	RX	4	nein	Glp	R1,D2(X2,B2)
78	LE	RX	4	nein	Glp	R1,D2(X2,B2)
79	CE	RX	4	ja	Glp	R1,D2(X2,B2)
7A	AE	RX	4	ja	Glp	R1,D2(X2,B2)
7B	SE	RX	4	ja	Glp	R1,D2(X2,B2)
7C	ME	RX	4	nein	Glp	R1,D2(X2,B2)
7D	DE	RX	4	nein	Glp	R1,D2(X2,B2)

Op. Code	Mnemo. Code	Bef. Typ	Länge in Byte	An-zeige	Bef. art	Assembler-format
7E	AU	RX	4	ja	Glp	R1,D2(X2,B2)
7F	SU	RX	4	ja	Glp	R1,D2(X2,B2)
86	BXH	RS	4	nein	Allg	R1,R3,D2(B2)
87	BXLE	RS	4	nein	Allg	R1,R3,D2(B2)
88	SRL	RS	4	nein	Allg	R1,D2(B2)
89	SLL	RS	4	nein	Allg	R1,D2(B2)
8A	SRA	RS	4	ja	Allg	R1,D2(B2)
8B	SLA	RS	4	ja	Allg	R1,D2(B2)
8C	SRDL	RS	4	nein	Allg	R1,D2(B2)
8D	SLDL	RS	4	nein	Allg	R1,D2(B2)
8E	SRDA	RS	4	ja	Allg	R1,D2(B2)
8F	SLDA	RS	4	ja	Allg	R1,D2(B2)
90	STM	RS	4	nein	Allg	R1,R3,D2(B2)
91	TM	SI	4	ja	Allg	D1(B1),I2
92	MVI	SI	4	nein	Allg	D1(B1),I2
93	TS	S	4	ja	Allg	D2(B2)
94	NI	SI	4	ja	Allg	D1(B1),I2
95	CLI	SI	4	ja	Allg	D1(B1),I2
96	OI	SI	4	ja	Allg	D1(B1),I2
97	XI	SI	4	ja	Allg	D1(B1),I2
98	LM	RS	4	nein	Allg	R1,R3,D2(B2)
9A	LAM	RS	4	nein	ESA	R1,R3,D2(B2)
9B	STAM	RS	4	nein	ESA	R1,R3,D2(B2)
AF	MC	SI	4	ja	Allg	D1(B1),I2
B205	STCK	S	4	ja	Allg	D2(B2)
B219	SAC	S	4	nein	ESA	D2(B2)
B222	IPM	RRE	4	nein	Allg	R1
B224	IAC	RRE	4	ja	ESA	R1
B22D	DXR	RRE	4	nein	Glp	R1,R2
B24C	TAR	RRE	4	ja	ESA	R1,R2
B24D	CPYA	RRE	4	nein	ESA	R1,R2
B24E	SAR	RRE	4	nein	ESA	R1,R2
B24F	EAR	RRE	4	nein	ESA	R1,R2
BA	CS	RS	4	ja	Allg	R1,R3,D2(B2)
BB	CDS	RS	4	ja	Allg	R1,R3,D2(B2)
BD	CLM	RS	4	ja	Allg	R1,M3,D2(B2)
BE	STCM	RS	4	nein	Allg	R1,M3,D2(B2)
BF	ICM	RS	4	ja	Allg	R1,M3,D2(B2)
D1	MVN	SS	6	nein	Allg	D1(L,B1),D2(B2)
D2	MVC	SS	6	nein	Allg	D1(L,B1),D2(B2)
D3	MVZ	SS	6	nein	Allg	D1(L,B1),D2(B2)
D4	NC	SS	6	ja	Allg	D1(L,B1),D2(B2)
D5	CLC	SS	6	ja	Allg	D1(L,B1),D2(B2)
D6	OC	SS	6	ja	Allg	D1(L,B1),D2(B2)
D7	XC	SS	6	ja	Allg	D1(L,B1),D2(B2)
DC	TR	SS	6	nein	Allg	D1(L,B1),D2(B2)
DD	TRT	SS	6	ja	Allg	D1(L,B1),D2(B2)

Op. Code	Mnemo. Code	Bef. Typ	Länge in Byte	An- zeige	Bef. art	Assembler- format
DE	ED	SS	6	ja	Dez	D1(L,B1),D2(B2)
DF	EDMK	SS	6	ja	Dez	D1(L,B1),D2(B2)
F0	SRP	SS	6	ja	Dez	D1(L1,B1),D2(B2),I3
F1	MVO	SS	6	nein	Allg	D1(L1,B1),D2(L2,B2)
F2	PACK	SS	6	nein	Allg	D1(L1,B1),D2(L2,B2)
F3	UNPK	SS	6	nein	Allg	D1(L1,B1),D2(L2,B2)
F8	ZAP	SS	6	ja	Dez	D1(L1,B1),D2(L2,B2)
F9	CP	SS	6	ja	Dez	D1(L1,B1),D2(L2,B2)
FA	AP	SS	6	ja	Dez	D1(L1,B1),D2(L2,B2)
FB	SP	SS	6	ja	Dez	D1(L1,B1),D2(L2,B2)
FC	MP	SS	6	nein	Dez	D1(L1,B1),D2(L2,B2)
FD	DP	SS	6	nein	Dez	D1(L1,B1),D2(L2,B2)

7.4 Erweiterter mnemotechnischer Operationscode

Zur Erleichterung für Programmierer verfügt der Assembler über erweiterte mnemotechnische Operationscodes. Sie ermöglichen es, bedingte Sprünge einschließlich ihrer Sprungmaske mnemotechnisch darzustellen. Der Assembler löst solche erweiterte mnemotechnische Operationscodes in die Befehle BC bzw. BCR auf und setzt die Maske.

Bei Befehlen mit zwei Bedeutungen, z.B. (Minus/Mixed), ist der Befehl mit der zweiten Bedeutung (Mixed) nach dem Befehl TM zu verwenden.

Assemblerformat mit erweitertem mnemotechnischem Operationscode		ergibt		Bedeutung
		Befehl	Maske,Operand	
B	D2(X2,B2)	BC	15,D2(X2,B2)	Branch
BE	D2(X2,B2)	BC	8,D2(X2,B2)	Branch when Equal
BH	D2(X2,B2)	BC	2,D2(X2,B2)	Branch when High
BL	D2(X2,B2)	BC	4,D2(X2,B2)	Branch when Low
BM	D2(X2,B2)	BC	4,D2(X2,B2)	Branch when Minus/Mixed
BNE	D2(X2,B2)	BC	7,D2(X2,B2)	Branch when Not Equal
BNH	D2(X2,B2)	BC	13,D2(X2,B2)	Branch when Not High
BNL	D2(X2,B2)	BC	11,D2(X2,B2)	Branch when Not Low
BNM	D2(X2,B2)	BC	11,D2(X2,B2)	Branch when Not Minus/Mixed
BNO	D2(X2,B2)	BC	14,D2(X2,B2)	Branch when Not Overflow/Ones
BNP	D2(X2,B2)	BC	13,D2(X2,B2)	Branch when Not Plus
BNZ	D2(X2,B2)	BC	7,D2(X2,B2)	Branch when Not Zero/Zeroes
BO	D2(X2,B2)	BC	1,D2(X2,B2)	Branch when Overflow/Ones
BP	D2(X2,B2)	BC	2,D2(X2,B2)	Branch when Plus
BR	R2	BCR	15,R2	Branch Register
BRE	R2	BCR	8,R2	Branch Register when Equal
BRH	R2	BCR	2,R2	Branch Register when High
BRL	R2	BCR	4,R2	Branch Register when Low
BRM	R2	BCR	4,R2	Branch Register when Minus/Mixed
BRNE	R2	BCR	7,R2	Branch Register when Not Equal
BRNH	R2	BCR	13,R2	Branch Register when Not High
BRNL	R2	BCR	11,R2	Branch Register when Not Low
BRNM	R2	BCR	11,R2	Branch Register when Not Minus/Mixed
BRNO	R2	BCR	14,R2	Branch Register when Not Overflow/Ones
BRNP	R2	BCR	13,R2	Branch Register when Not Plus
BRNZ	R2	BCR	7,R2	Branch Register when Not Zero/Zeroes
BRO	R2	BCR	1,R2	Branch Register when Overflow/Ones
BRP	R2	BCR	2,R2	Branch Register when Plus
BRZ	R2	BCR	8,R2	Branch Register when Zero/Zeroes
BZ	D2(X2,B2)	BC	8,D2(X2,B2)	Branch when Zero/Zeroes
NOP	D2(X2,B2)	BC	0,D2(X2,B2)	No Operation
NOPR	R2	BCR	0,R2	No Operation Register

7.5 Zweierpotenzen

Wert	Dezimale Darstellung	Sedezimale Darstellung	
2^0	1	1	
2^1	2	2	
2^2	4	4	
2^3	8	8	
2^4	16	10	
2^5	32	20	
2^6	64	40	
2^7	128	80	
2^8	256	1 00	
2^9	512	2 00	
2^{10}	1 024	4 00	
2^{11}	2 048	8 00	
2^{12}	4 096	10 00	
2^{13}	8 192	20 00	
2^{14}	16 384	40 00	
$2^{15}-1$	32 767	7F FF	
2^{15}	32 768	80 00	
$2^{16}-1$	65 535	FF FF	
2^{16}	65 536	1 00 00	
2^{17}	131 072	2 00 00	
2^{18}	262 144	4 00 00	
2^{19}	524 288	8 00 00	
2^{20}	1 048 576	10 00 00	
2^{21}	2 097 152	20 00 00	
2^{22}	4 194 304	40 00 00	
2^{23}	8 388 608	80 00 00	
$2^{24}-1$	16 777 215	FF FF FF	
2^{24}	16 777 216	1 00 00 00	
2^{25}	33 544 432	2 00 00 00	
2^{26}	67 108 864	4 00 00 00	
2^{27}	134 217 728	8 00 00 00	
2^{28}	268 435 456	10 00 00 00	
2^{29}	536 870 912	20 00 00 00	
2^{30}	1 073 741 824	40 00 00 00	
$2^{31}-1$	2 147 483 647	7F FF FF FF	
2^{31}	2 147 483 648	80 00 00 00	*)
$2^{32}-1$	4 294 976 295	FF FF FF FF	*)
-1	-1	FF FF FF FF	
-2	-2	FF FF FF FE	
-2^{15}	-32 768	FF FF 80 00	
-2^{31}	-2 147 483 648	80 00 00 00	

*) Vorzeichenlose 32 Bit lange Binärzahl.

7.6 Zugriff auf gemeinsam benutzte Daten in Multiprozessor-Anlagen

Der konkurrierende Zugriff mehrerer Programme (Tasks/Contingency-Prozesse) auf gemeinsame Daten im Hauptspeicher muß wegen der Möglichkeit simultaner Speicherzugriffe in Multiprozessor-Anlagen bzw. wegen der Unterbrechbarkeit sorgfältig programmiert werden. Sowohl schreibende als auch lesende Zugriffe auf gemeinsam benutzte Daten sind mit einer **Sperre** (Lock) gegen eine Verfälschung vor konkurrierenden Schreibzugriffen zu schützen (ein Datum kann auch selbst als Sperre behandelt werden).
Diese Sperre kann ein Byte, ein Wort oder ein Doppelwort lang sein.
Eine **binäre Sperre** von einem Wort Länge (auch Lock-Wort genannt) kann entweder 0 für 'nicht belegt' oder einen anderen Wert wie X'FFFFFFFF' für 'belegt' enthalten.
In einer **Zähler-Sperre** wird ein Zähler herauf- oder heruntergezählt.

Im folgenden werden die Begriffe **'sicheres Lesen'**, **'sicheres Schreiben'**, **'sichere Befehle'** oder **'sichere Operation'** verwendet, das heißt:
Kein anderer Prozessor kann während des Schreib- oder Lesevorgangs ein Byte innerhalb des Wortes oder Doppelwortes oder ein Bit innerhalb des Byte oder Wortes oder Doppelwortes verändern.

Nur bestimmte Befehle sind 'sichere Befehle' und damit zum Setzen oder Rücksetzen oder Abfragen von Sperren geeignet.
Absolut **sichere Befehle** auf allen Anlagen sind: **CS, CDS und TS**; siehe Kapitel 3 und Abschnitt 7.6.1
Allerdings benötigen diese Befehle die 10-fache Ausführungszeit wie z.B. ein ST, auf manchen Anlagentypen sogar noch mehr.

Das **Rücksetzen** eines **Byte** mit **MVI** gilt als 'sichere Operation'.
Das **Rücksetzen** eines **Wortes** oder **Doppelwortes** mit den Befehlen **MVC, ST, STM, STD** gilt erst ab der Zentraleinheit 7.590 als 'sichere Operation'. Bei abwärtskompatiblen Zentraleinheiten (≤ 7.580) muß das Rücksetzen von Sperren oder das Löschen von (nicht durch eine Sperre geschützten) gemeinsam benutzten Speicherbereichen mit mehr als 1 Byte Länge (d.h. Wort- oder Doppelwortlänge) mit den Befehlen CS oder CDS ausgeführt werden.

Das **Abfragen** eines **Byte** mit **CLI** gilt als 'sichere Operation'.
Das **Abfragen** eines **Wortes** oder **Doppelwortes** mit den Befehlen **C, CL, CLC, IC, L, LM, LD, MVC** gilt erst ab der Zentraleinheit 7.590 als 'sichere Operation'. Bei abwärtskompatiblen Zentraleinheiten (≤ 7.580) muß das Abfragen von Sperren oder von (nicht durch eine Sperre geschützten) gemeinsam benutzten Speicherbereichen mit mehr als 1 Byte Länge (d.h. Wort- oder Doppelwortlänge) wiederholt werden, um sicherzustellen, daß sie keinen verfälschten Wert gelesen haben.

7.6.1 Setzen von Sperren

Folgende Befehle eignen sich zum Setzen von Sperren:

TS (1 Byte, allerdings nur 2 Werte, = X’FF’ und ≠ X’FF’,
daher nur für binäre Sperren geeignet)
CS (1 Wort)
CDS (1 Doppelwort)

Bei diesen Befehlen ist gewährleistet, daß während des Update-Vorgangs ein simultaner Zugriff eines anderen Prozessors auf den gleichen Speicherplatz ausgeschlossen ist. Der ursprüngliche Wert der entsprechenden Operanden dieser Befehle kann über die gesetzte Anzeige abgefragt werden.

Nicht geeignet sind folgende Befehle und alle anderen Speicherzugriffsbefehle, da sie einen simultanen Zugriff eines anderen Prozessors nicht ausschließen:

MVI, NI, OI, XI, MVC, NC, OC, XC, ST, STM, STC.

7.6.2 Rücksetzen von Sperren

Zum Rücksetzen von binären Sperren auf Rechnern ≥ 7.590 sind folgende Befehle geeignet:

- für ein Byte der Befehl MVI
- für ein Wort die Befehle ST und MVC
- für ein Doppelwort die Befehle STM, STD und MVC

Wie bereits weiter vorn erwähnt, sind auf anderen Rechnern die Befehle CS, CDS und TS zu verwenden. Da das Schreiben bei diesen Befehlen für die gegebenen Längen sicher ist, liegt in Kombination mit TS/CS/CDS immer eine eindeutige Speicherbelegung vor.

Bedingungen bei Verwendung des Befehls MVC sind:
die beiden Operanden dürfen sich nicht überlappen, und beide müssen auf Wort- oder Doppelwort-Grenze (entsprechend ihrer Länge) ausgerichtet sein.

Bei Zähler-Sperren ist das Herunterzählen als Setzen zu betrachten, also unbedingt mit den Befehlen CS/CDS ausführen.

Nicht geeignet zum Zurücksetzen von binären Sperren und Lock-Worten sind die nachfolgend aufgeführten Befehle, bei denen zwei Speicherzugriffe notwendig sind (erst Lesen, dann Schreiben), weil sie nicht gegen dazwischenkommende Zugriffe von anderen Prozessoren geschützt sind:

NI, NC, OI, OC, XI, XC.

7.6.3 Abfragen von Sperren

Zum Abfragen von Sperren sind auf Anlagen ≥ 7.590 folgende Befehle geeignet:

- für ein Byte die Befehle CLI und IC
- für ein Wort die Befehle C, CL, CLC, L, MVC
- für ein Doppelwort die Befehle LM, LD, CLC, MVC

Wie bereits weiter vorn erwähnt, sind auf anderen Rechnern die Befehle (außer dem CLI) zu wiederholen.

Bedingungen bei Verwendung des Befehls CLC sind:
die beiden Operanden dürfen sich nicht überlappen, und beide müssen auf Wort- oder Doppelwort-Grenze (entsprechend ihrer Länge) ausgerichtet sein.

Nicht geeignet zum Abfragen von Sperren sind die nachfolgend aufgeführten Befehle, bei denen zwei Speicherzugriffe notwendig sind (erst Lesen, dann Schreiben), weil sie nicht gegen dazwischenkommende Zugriffe von anderen Prozessoren geschützt sind:

NI, NC, OI, OC, XI, XC, ZAP.

7.6.4 Beispiele

Zähler-Sperren

Beispiel 1: Hochzählen

```
           L     Rold,SPERRE
   ┌────── @CYCLE
   │         LR    Rnew,Rold
   │         AH    Rnew,=H’1’
   ├────── @WHEN EQ
   │         CS    Rold,Rnew,SPERRE
   ├──←─── @BREAK
   └────── @BEND
```

Beispiel 2: Bedingtes Hochzählen

```
   ┌────── @CYCLE
   │         L     Rold,SPERRE
   ├────── @WHEN GZ
   │         LTR   Rnew,Rold
   ├────── @AND  EQ
   │         AH    Rnew,=H’1’
   │         CS    Rold,Rnew,SPERRE
  ───←──── @BREAK
   │  ┌────  @IF   LE
   │  │        LTR    Rold,Rold
   │  ├────  @THEN
   │  │        @PASS NAME=wait
   │  └────  @BEND
   └────── @BEND
```

Falsch wäre ein zweiter Zugriff auf den Wert der Sperre (z.B. `L Rnew,SPERRE` statt `LTR Rnew,Rold`), weil dann nicht mehr garantiert wäre, daß der modifizierte Wert und der Vergleichswert sich genau um die beabsichtigte Differenz unterscheiden - die Sperre kann zwischen zwei Zugriffsbefehlen von einem anderen Programm (Task/Contingency) verändert worden sein !

Die Zuweisung mit dem Befehl `L Rold,SPERRE` muß im zweiten Beispiel innerhalb der Schleife stattfinden, weil der CS-Befehl nur dann ausgeführt wird, wenn der erste Teil der Abfrage (Rold > 0) wahr ist.

Betriebsmittelverwaltung

Es sollen gleichartige Betriebsmittel verwaltet werden, z.B. gleich große Teile eines Speicherbereiches, die Elemente (oder Einträge) heißen. Ein solches Element kann bei Bedarf für Tabellen dynamisch zugeordnet und wieder freigegeben werden. Jedes Element kann demnach entweder ’frei’ oder ’belegt’ sein. Für die Verwaltung der Elemente existiert ein Bitvektor, in dem jedes Bit einem Element zugeordnet ist und dessen Zustand kennzeichnet:

Bit(N) = 0 bedeute ’Element(N) ist frei’
Bit(N) = 1 bedeute ’Element(N) ist belegt’

Der Einfachheit halber nehmen wir an, daß maximal 64 Elemente und damit 64 Bits für unsere Zwecke ausreichen.

Beispiel 3: Belegen eines Elements

```
SPERRE DC     D’0’
       ...
OLD    DS     D
NEW    DS     D


       ...
       SLR    R4,R4
       LA     R5,1
       SLDL   R4,64-bitnum (bitnum 1...64)
       LM     R6,R7,SPERRE
           @CYCLE
             LR    R8,R6
             LR    R9,R7
           @IF   NZ
             NR    R8,R4
           @OR   NZ
             NR    R9,R5
           @THEN
             LA    R15,schon_belegt
             @EXIT

           @BEND
           LR   R8,R6
           LR   R9,R7
           OR   R8,R4
           OR   R9,R5
         @WHEN EQ
           CDS  R6,R8,SPERRE

           @BREAK
           @BEND
```

Das Registerpaar (R4,R5) enthält ein auf 1 gesetztes Bit an der Stelle, die das zu belegende Element repräsentiert; alle anderen Bits sind 0. Das Registerpaar (R6,R7) enthält den alten Inhalt des Bitvektors ’SPERRE’. Das Registerpaar (R8,R9) entsteht aus der ODER-Verknüpfung von (R6,R7) mit (R4,R5), hat also alle in (R6,R7) gesetzten Bits und zusätzlich das in (R4,R5) gesetzte Bit gesetzt. Das Ergebnis der Verknüpfung wird geschützt mittels des CDS-Befehls in den Bitvektor ’SPERRE’ eingetragen.

Beispiel 4: Freigeben eines Elements (Gegenbeispiel)

```
SPERRE DC      D'0'
        ...
OLD    DS      D
NEW    DS      D


      ...
      SLR      R4,R4
      LA       R5,1
      SLDL     R4,64-bitnum  (bitnum 1..64)
      X        R4,=A(X'FFFFFFFF')
      X        R5,=A(X'FFFFFFFF')
      LM       R8,R9,SPERRE        *
 /\/→                              *
      LM       R6,R7,SPERRE        *
      ┌─── @CYCLE                  *
      │      NR    R8,R4           *
      │      NR    R9,R5           *
      ├─── @WHEN EQ
      │      CDS   R6,R8,SPERRE
      │
      ├─←  @BREAK
      │      LR    R8,R6           @
      │      LR    R9,R7           @
      └─── @BEND
```

Das Registerpaar (R4,R5) enthält ein auf 0 gesetztes Bit an der Stelle, die das freizugebende Element repräsentiert; alle anderen Bits sind 1. Das Registerpaar (R6,R7) enthält den alten Inhalt des Bitvektors ’SPERRE’. Das Registerpaar (R8,R9) entsteht aus der UND-Verknüpfung von (R6,R7) mit (R4,R5), hat also genau die in (R6,R7) gesetzten Bits ohne das in (R4,R5) gesetzte Bit gesetzt. Das Ergebnis der Verknüpfung wird geschützt mittels des CDS-Befehls in den Bitvektor eingetragen.

Der Fehler liegt darin, daß der Vergleichsinhalt (R6,R7) und der Basisinhalt für die Modifikation (R8,R9) in zwei Schritten aus dem Speicher geholt wird. Findet an der durch `/\/→` gekennzeichneten Stelle eine Programmunterbrechung statt, in der das Doppelwort ’SPERRE’ verändert wird, oder es wird an dieser Stelle durch einen anderen Prozessor verändert, dann wird diese Änderung wieder rückgängig gemacht. Ein gerade von einem anderen Programm (Prozeß/Task) freigegebenes Element wird als belegt gekennzeichnet, oder (noch schlimmer!) ein gerade von einem anderen Programm belegtes Element wird ungerechtfertigt freigegeben. Auch ein Vertauschen der beiden Befehle vor und nach der mit `/\/→` gekennzeichneten Stelle würde ein Fehlverhalten nur unwahrscheinlicher machen, aber nicht ausschließen.

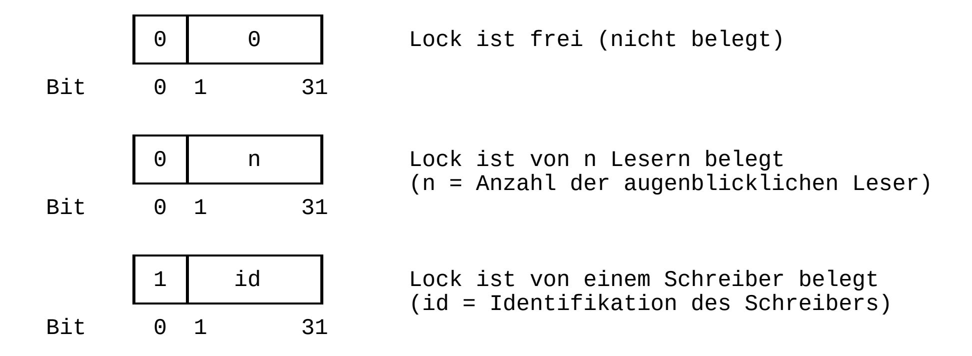
Eine Lösung muß auf den zweiten Zugriff auf SPERRE verzichten und (R8,R9) wie oben im Beispiel 3 am Anfang der Schleife aus (R6,R7) initialisieren:

```
LM     R6,R7,SPERRE
┌─ @CYCLE
│    LR   R8,R6
│    LR   R9,R7
│    NR   R8,R4
│    NR   R9,R5
```

Mit diesem Code ist die erste oben mit * gekennzeichnete Codefolge zu ersetzen, die zweite mit @ gekennzeichnete Codefolge kann dann entfallen.

Reader-Writer-Synchronisation

Hierfür wird im einfachsten Fall ein Lock-Wort benötigt, das z.B. folgende Zustände annehmen kann:



Ein Leser muß folgendes Protokoll einhalten:

```
Lese-Lock anfordern (GET␣READ␣LOCK)
Lesen
Lese-Lock freigeben (REL␣READ␣LOCK)
```

Ein Schreiber muß folgendes Protokoll einhalten:

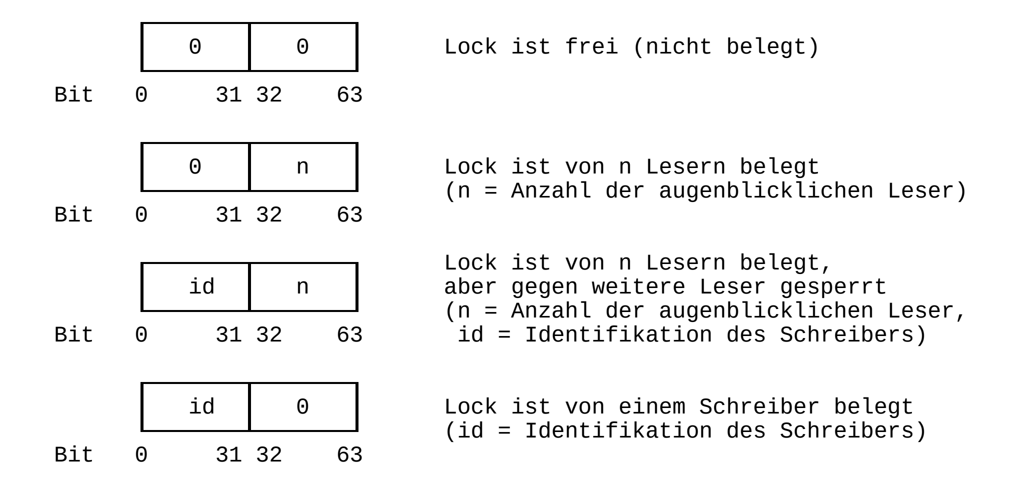
```
Schreib-Lock anfordern (GET␣WRITE␣LOCK)
Schreiben
Schreib-Lock freigeben (REL␣WRITE␣LOCK)
```

In der folgenden Tabelle ist für jede der benötigten Lock-Funktionen aufgeführt, welche Zustandsübergänge sie für das Lock-Wort durchführen, und ob dafür ein CS-Befehl notwendig ist.

Funktion	Zustandsübergang	CS
GET␣READ␣LOCK	(0,n) -> (0,n+1)	ja
REL␣READ␣LOCK	(0,n) -> (0,n-1) Bedingung: n>0, sonst Fehler	ja
GET␣WRITE␣LOCK	(0,0) -> (1,id)	ja
REL␣WRITE␣LOCK	(1,id) -> (0,0)	nein

Diese Art der Synchronisation ist nicht effizient, weil ein Schreiber möglicherweise niemals sein Schreib-Lock bekommt und deshalb nie an die Reihe kommt. Sie ist also nur dann anzuwenden, wenn die möglichen Leser nur selten ein Lese-Lock anfordern und es nicht lange halten.

Ein effizientes Synchronisationsverfahren läßt sich z.B. mit einem Lock-Doppelwort realisieren, das folgende Zustände annehmen kann:



Ein Leser muß folgendes Protokoll einhalten:

```
Lese-Lock anfordern (GET␣READ␣LOCK)
Lesen
Lese-Lock freigeben (REL␣READ␣LOCK)
```

Ein Schreiber muß folgendes Protokoll einhalten:

```
Schreib-Lock anfordern und Lese-Lock sperren  (GET␣WRITE␣LOCK␣INHIBIT␣READ)
Warten auf Freigabe des Lese-Locks
Schreiben
Schreib-Lock freigeben und Lese-Sperre aufheben (REL␣WRITE␣LOCK␣ADMIT␣READ)
```

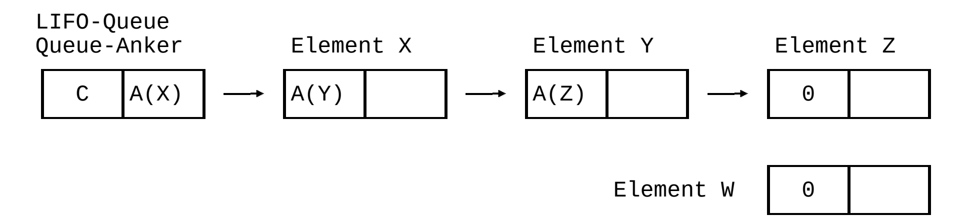
In der folgenden Tabelle ist für jede der benötigten Lock-Funktionen aufgeführt, welche Zustandsübergänge sie für das Lock-Wort durchführen, und ob dafür ein CDS-Befehl oder ein CS-Befehl notwendig ist.

Funktion	Zustandsübergang	CS/CDS
GET␣READ␣LOCK	(0,n) -> (0,n+1)	CDS
REL␣READ␣LOCK	(x,n) -> (x,n-1) x = 0 | id Bedingung: n>0, sonst Fehler	CS lock.rechtes␣wort
GET␣WRITE␣LOCK␣...	(0,n) -> (id,n)	CS lock.linkes␣wort
REL␣WRITE␣LOCK␣...	(id,0) -> (0,0)	nein

Multiprozessor-Fähigkeit von Warteschlangen-Mechanismen

Anhand der Ein- und Auskettung einer LIFO(last in first out)-Warteschlange (Queue) wird die Multiprozessor-Fähigkeit dargestellt.

Vorgegeben sei eine Warteschlange mit Anker Q und den eingeketteten Elementen X, Y und Z. Das freie Element W soll zu geeigneter Zeit eingekettet werden.



Möchten Sie bei der Bearbeitung der Warteschlange auf den Schutz durch eine Sperre verzichten (weil hierdurch zu große Performance-Einbußen entstehen würden), müssen Sie für den Update der Warteschlange den CS-Befehl benutzen.

Für die Einkettung eines Elements in die Warteschlange reicht der CS-Befehl aus, um die Serialisierung von Prozessen, welche möglicherweise gleichzeitig einketten wollen, zu garantieren. Dies ergibt sich aus der Tatsache, daß für die Übernahme des Zeigers auf das vorherige vorderste Element in das Link-Feld des einzukettenden Elements und den Update des Warteschlangen-Ankers nur dieser ausgewertet werden muß.

Beim Ausketten reicht dagegen der CS-Befehl nicht mehr aus, um den gleichzeitigen Zugriff mehrerer Ausketter auszuschließen. Man betrachte hierzu folgenden Ablauf auf der oben dargestellten Warteschlangen-Konfiguration:

Ausketten durch Task/Prozessor 1	Andere Tasks/ Prozessoren	Zustand der Warteschlange
Ausgangslage	:	Q → X → Y → Z
:	:	
L Ra,A(X) 1. Element	:	
L Rc,A(Y) verk. Element	:	
:	:	
:	Ausketten Element X	Q → Y → Z
:	Ausketten Element Y	Q → Z
:	Einketten Element W	Q → W → Z
:	Einketten Element X	Q → X → W → Z
:	:	
CS Ra,Rc,Queue-Anker	:	Q → Y → ?
:	:	
Element X ausgekettet	Ausketten Element Y	Q → ?

Task/Prozessor 1 beginnt die Auskettung des Elements X. Nachdem die Adresse des auszukettenden Elements sowie der Link zum nächsten Element geladen sind, wird Task/Prozessor 1 verzögert (z.B. wegen Taskwechsel, Ausbremsung des Prozessors, Eingriff durch VM2000). Bevor der Update der Warteschlange mithilfe des CS-Befehls vollzogen werden kann, erfolgen weitere Warteschlangen-Aktionen durch andere Tasks/Prozessoren.

Wird erst danach der CS-Befehl von Task/Prozessor 1 wirksam, wird bei obiger Konstellation die erfolgreiche Ausführung angezeigt, weil sich der Inhalt des Warteschlangen-Ankers gegenüber dem vorausgehenden Ladebefehl nicht verändert hat. Der Anker wurde mit dem vermeintlich noch verketteten Element Y versorgt. Das in der Unterbrechung eingekettete Element W geht zusammen mit den nachfolgenden Warteschlangen-Elementen verloren.

Folgt im Anschluß an die bisherige Warteschlangen-Folge eine weitere Auskettung, so wird das Element Y ein zweites Mal vergeben. Es wird damit von zwei Instanzen parallel, aber ohne gegenseitige Koordination verwendet. Sich hieraus möglicherweise ergebende Fehler sind Überschreiber und Ring-Verkettungen (wenn z.B. das Element Y zweimal in die obige Warteschlange zurückgekettet wird).

Diese Fehler sind - wenn überhaupt - sehr schwer diagnostizierbar, da sie sich meist erst zu einem sehr viel späteren Zeitpunkt bemerkbar machen. Abhilfe schafft hier ein Zähler, um welchen der Warteschlangen-Anker erweitert wird. Dieser darf nur in eine Richtung (z.B. aufwärts) gezählt werden. Anstelle des CS-Befehls ist dann der CDS-Befehl zu verwenden, mit welchem der Warteschlangen-Anker in der Länge von zwei Worten multiprozessor-fest modifiziert werden kann. Der obige Algorithmus ist entsprechend der obigen Abbildung wie folgt anzupassen:

```
L    Ra,C              Lade aktuellen Zähler aus Warteschlangen-Anker
L    Rb,A(X)           Lokalisiere Adresse des vordersten Elements
LA   Rc,1(,Ra)         Erhöhe Zähler (Wrap Around)
L    Rd,A(Y)           Lokalisiere Adresse des verketteten Elements

CDS Ra,Rc,Queue-Anker     Führe Auskettung aus
```

Ein Zähler, welcher stets die Anzahl der in der Warteschlange befindlichen Elemente anzeigt, ist nicht geeignet, denn er würde im obigen Warteschlangen-Beispiel genauso unverändert erscheinen wie das im Anker angezeigte Warteschlangen-Element.

Literatur

[1] **ASSEMBH** (BS2000)
Beschreibung

Zielgruppe
Anwender, die im BS2000 Programme in der Assembler- oder Makrosprache erstellen sowie die strukturierte Programmierung benutzen wollen
Inhalt
- Beschreibung des Sprachumfangs des Assemblers ASSEMBH im BS2000
- Struktur der Assemblersprache, Assembleranweisungen
- Struktur und Elemente sowie Instruktionen der Makrosprache
- Strukturierte Programmierung mit ASSEMBH
- vordefinierte Makros für die strukturierte Programmierung
- ILCS-Anschluß für die strukturierte Programmierung

[2] **Einführung in die XS-Programmierung**
(für ASSEMBLER-Programmierer) (BS2000)
Benutzerhandbuch

Zielgruppe
- Programmierer
- System-Programmierer

Inhalt
- Die Adressierungsmodi der ab BS2000 V9.0 unterstützten Zentraleinheiten und wie sie sich auf die ASSEMBLER-Befehle auswirken
- Programmierhinweise für ASSEMBLER-Programmierer, die den erweiterten Adreßraum von XS-Anlagen nutzen oder ihre Programme adressierungsmodus-unabhängig und portabel gestalten wollen;

Einsatz
TU-, TPR-Programme

[3] **BS2000/OSD-BC V1.0**
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an alle BS2000/OSD-Assembler-Programmierer.
Inhalt
Das Handbuch enthält eine Zusammenstellung der Makroaufrufe an den Ablaufteil, die ausführliche Beschreibung jedes Makroaufrufs mit Hinweisen und Beispielen, einschließlich der Jobvariablen-Makros, sowie einen ausführlichen allgemeinen Lernteil.

[4] **BS2000/OSD-BC V1.0**
Systemanwendung
Taschenbuch

Zielgruppe
Erfahrene BS2000-Anwender
Inhalt
Enthält in Form von Auszügen aus anderen Handbüchern konzentrierte Informationen zum Betrieb des BS2000.
- Befehle und Anweisungen des ASSEMBH
- BS2000 Makros und Kommandos
- Anweisungen zu AID, DAMP, ARCHIVE, SM2-PA, SDF-I, PASSWORD, DPAGE, TPCOMP2, SPCCNTRL, PAMCONV, MSGEDIT, MSGLIB, FDEXIM, PRSERVE, SPOOLSERVE, RSOSERVE
- Auftragsschalter, Dateikettungsnamen und Code Tabellen des BS2000.

Einsatz
BS2000-Dialogbetrieb und -Stapelbetrieb.

Bestellen von Handbüchern

Die aufgeführten Handbücher finden Sie mit ihren Bestellnummern im *Druckschriftenverzeichnis* der Siemens Nixdorf Informationssysteme AG. Neu erschienene Titel finden Sie in den *Druckschriften-Neuerscheinungen*.

Beide Veröffentlichungen erhalten Sie regelmäßig, wenn Sie in den entsprechenden Verteiler aufgenommen sind. Wenden Sie sich bitte hierfür an Ihre zuständige Geschäftsstelle. Dort können Sie auch die Handbücher bestellen.

Stichwörter

Inhalt

Assemblerbefehle (BS2000/OSD)

Sprachbeschreibung

Zielgruppe

BS2000/OSD-Assembler-Programmierer

Inhalt

Beschrieben sind alle Assemblerbefehle (nicht privilegiert) der vom BS2000/OSD unterstützten Zentraleinheiten in alphabetischer Reihenfolge.
Bei jedem Befehl sind dargestellt:
- seine Funktion
- sein Assemblerformat, d.h. seine Schreibweise in Assemblersprache
- sein Maschinenformat, d.h. seine Darstellung in der Zentraleinheit
- sein Ablauf im Detail
- etwaige von ihm gesetzte Werte der Anzeige
- die bei seinem Ablauf möglichen Programmunterbrechungen
- Programmierhinweise
- ein oder mehrere Beispiele

Ausgabe: **Mai 1993**

Datei: **ASSEMBL.PDF**

